University of Alberta

Empirically Driven Investigation of Dependability and Security Issues in Internet-Centric Systems

by

Huynh, Toan Nguyen Duc

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

in

Software Engineering and Intelligent Systems

Department of Electrical and Computer Engineering

Spring 2010
Edmonton, Alberta

Examining Committee

Dr. James Miller, Department of Electrical and Computer Engineering

Dr. Yu (Bryan) Hu, Department of Electrical and Computer Engineering

Dr. Vincent Gaudet, Department of Electrical and Computer Engineering

Dr. H. James Hoover, Department of Computing Science

Dr. John Aycock, Department of Computer Science, University of Calgary

Dedicated to my parents Long Huỳnh and Nữ Nguyễn.

You give everything and expect nothing in return.
I am eternally grateful for all you have done.

***

Công Cha như núi Thái Sơn,
Nghĩa Mẹ như nước trong nguồn chảy ra.

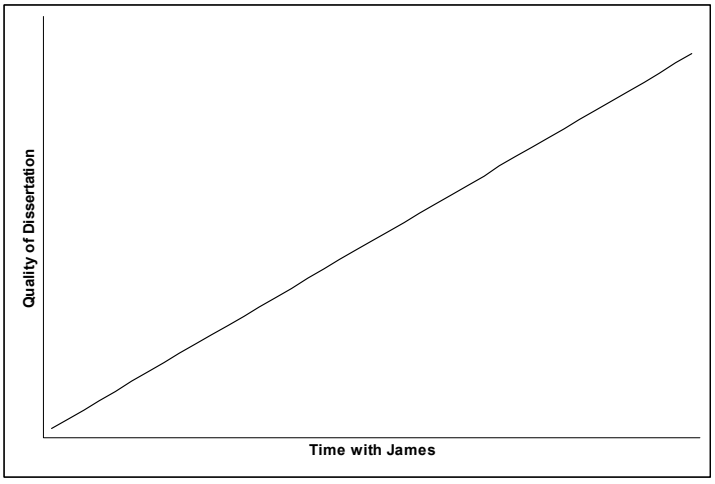
Abstract

The Web, being the most popular component of the Internet, has been transformed from a static information-serving medium into a fully interactive platform. This platform has been used by developers to create web applications rivaling traditional desktop systems. Designing, developing and evaluating these applications require new or modified methodologies, techniques and tools because of the different characteristics they exhibit. This dissertation discusses two important areas for developing and evaluating these applications: security and data mining.

In the security area, a survey using a process similar to the Goal Question Metric approach examines the properties of web application vulnerabilities. Using results from the survey, a white-box approach to identify web applications' vulnerabilities is proposed. Although the approach eliminates vulnerabilities during the development process, it does not protect existing web applications that have not utilized the approach. Hence, an Anomaly-based Network Intrusion Detection System, called AIWAS, is introduced. AIWAS protects web applications through the analysis of interactions between the users and the web applications. These interactions are classified as either benign or malicious; malicious interactions are prevented from reaching the web applications under protection.

In the data mining area, the method of reliability estimation from server logs is examined in detail. This examination reveals the fact that the session workload is currently obtained using a constant Session Timeout Threshold (STT) value. However, each website is unique and should have its own STT value. Hence, an initial model for estimating the STT is introduced to encourage future research on sessions to use a customized STT value per website. This research on the STT leads to a deeper investigation of the actual session workload unit. More specifically, the distributional properties of the session workload are re-examined to determine whether the session workload can be described as a heavy-tailed distribution.

Acknowledgement



As you can see from the graph, without my supervisor Dr. James Miller, this dissertation simply does not exist. I am grateful for his valuable guidance and criticism; his extreme patience should also be commended. Dr James Miller is not only an excellent mentor for my research; he's also a friend. I will always be thankful for all the assistance he has provided. Thank you James.

Special thanks to John Bringas, P.Eng., for his constant support and encouragement. His out-of-the-box questions have always kept me on my feet. Even before I finished my undergraduate degree, he has encouraged me to pursue higher education. His encouragement is one of the reasons why I am here today. Not only that, he started getting me involved with running which helped keep me in shape despite the hours I toiled away in front of the computer. For all his support – thank you John.

I would also like to thank my family who always tried to make sure I am well nourished and healthy. To my friends who are a constant source of entertainment, and to my colleagues who had to cover for me whenever I mysteriously disappear for a few hours.

Funding for this research was provided by the Natural Sciences and Engineering Research Council of Canada (NSERC) and *CASTI*.

Table of Contents

List of Tables

List of Figures

List of Acronyms

A

ACF: Autocorrelation Function

A-NIDS: Anomaly-based Network Intrusion Detection System

AJAX: Asynchronous JavaScript and XML

API: Application Programming Interface

C

CERIAS: Center for Education and Research in Information Assurance and Security

CDG: Contamination Data Graph

CG: Contamination Graph

CLF: Common Log Format

CMS: Content Management System

COM: Component

COTS: Commercial off the Shelf

CVS: Concurrent Versions System

D

DBMS: Database Management System

E

EGPCS: Environment, GET, POST, Cookie, Server

EIV: External Interaction Vulnerability

EOP: End of Program

ES: External Sources

F

FPR: False Positive Rate

FTP: File Transfer Protocol

G

GB: Gigabyte

H

HTTP: Hypertext Transfer Protocol

I

IDS: Intrusion Detection System

IE: Internet Explorer

IIS: Internet Information Services

IP: Internet Protocol

IM: Instance Model

IMME: Instance Model Mapping Engine

K

KLOC: Thousands of Lines of Code

L

LLCD: Log Log Complementary Distribution

loc: location

LOC: Lines of Code

M

MCC: Matthew's Correlation Coefficient

ML: Machine Learning

MTBF: Mean Time between Failure

MWBF: Mean Workload between Failure

N

NIDS: Network Intrusion Detection System

NXD: Native XML Database

O

OS: Operating System

OSVDB: Open Source Vulnerability Database

Q

Q-Q Plot: Quantile Quantile Plot

R

RFC: Request for Comment

ROC: Receiver Operating Characteristics

S

SCF: Source Content Failure

SDG: System Dependency Graph

SPSS: Statistical Package for the Social Sciences

SQL: Structured Query Language

STT: Session Timeout Threshold

T

TPR: True Positive Rate

U

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

US-CERT: United States Computer Emergency Readiness Team

V

VDB: Vulnerability Database

W

WAIC: Web Application Input Collection

WAGG: Web Application Graph Generation

X

XPATH: XML Path Language

XRF: Cross-site Request Forgery

XSS: Cross-site Scripting

Chapter 1- Introduction

The Web was introduced to the Internet in 1991. Within 19 years, it has transformed from a medium used to present information statically to a modern medium capable of e-commerce, entertainment, surveys, and many other activities. In fact, web applications are now a crucial component for many companies. These applications are now one of the most important parts of the software industry. Yet, they have different characteristics that make them different from traditional software and information systems. For example, web applications have short release cycles and development time (Baskerville and Pries-Heje 2004). Many new features, enhancements and bug fixes are continually added during these cycles. Furthermore, developers often build web applications by integrating many existing parts together. For example, a legacy system can be combined with several Commercial off the Shelf (COTS) packages by custom in-house code to create a complete web application. The source code for the COTS packages is often unavailable to the developers or these COTS packages may exist on remote servers as web services. When coupled with web applications' ability to transfer data among completely different types of components, web applications must now ensure data is persistent through user sessions, across sessions, and shared among sessions. As a result, web engineering is a recent field that focuses on the methodologies, techniques and tools to design, develop, and evaluate web applications. This dissertation contributes to this field in two areas: security and data mining.

1.1 Web Application Security

The Laws of Vulnerabilities 2.0[1] states that "80 percent of vulnerability exploits are now available within single digit days after the vulnerability's public release". The 2008 Internet Security Threat Report[2] from Symantec notes that web applications contain 63 percent of all documented vulnerabilities. Insecure applications can be extremely costly. For example, ChoicePoint, after exposing 145,000 customer accounts, reported $11.4 million in charges directly related to the incident (Rapid7 2005). Immediately after the incident was disclosed, ChoicePoint's total market capitalization dropped by $720 million. Meanwhile, CardSystems is barred from accepting Visa and American Express cards after compromising 40 million accounts due to a SQL Injection vulnerability. Hence, security is a prominent non-functional requirement for modern web applications.

Chapters 2-4 explore this prominent non-functional requirement in detail. To begin, Chapter 2 performs a survey on the vulnerabilities using a method similar to the Goal Question Metric (GQM) approach. Four questions are raised.

1. What proportion of security vulnerabilities in web applications can be considered as implementation vulnerabilities?

[1] http://www.qualys.com/research/rnd/vulnlaws/, last accessed August 16, 2009
[2] http://www4.symantec.com/Vrt/wl?tu_id=gCGG123913789453640802, last accessed January 29, 2010

2. Are these vulnerabilities the result of interactions between web applications and external systems?
3. What is the proportion of vulnerable LOC within a web application?
4. Are implementation vulnerabilities caused by implicit or explicit data flows?

The results obtained show that the majority of web application vulnerabilities are of the implementation type which is caused by insecure coding practices.

Based on the information from Chapter 2, Chapter 3 introduces a practical approach to eliminate web vulnerabilities. Through effective use of computer-support software to automate the "straightforward" components, the approach enables the security tester to concentrate on the "creative" component in vulnerability detection. Furthermore, this approach integrates into the software development process. This integration allows software development organizations to identify and eliminate the vulnerabilities before the product is shipped or launched.

Although the approach presented in Chapter 3 allows vulnerabilities to be removed, it does not allow web administrators to protect their pre-existing platforms against attacks. Hence, Chapter 4 presents an Anomaly-based Network Intrusion Detection System (A-NIDS), called AIWAS, to guard web applications from malicious users. Instead of removing vulnerabilities from web applications, AIWAS classifies behaviours from users as either benign or malicious. It does this by learning from the input which is the primary way for users to interact with web applications. Essentially, AIWIAS studies the input specification associated with "normal" usage of the system and validates any given inputs against this specification. This technique allows AIWAS to filter out malicious inputs before they reach the web application.

1.2 Data Mining Web Server Logs

The World Wide Web is now the most popular component of the Internet (Arlitt and Williamson 1997). The Web can be utilized for many purposes ranging from information retrieval to fully interactive e-commerce stores. Companies increasingly use the Web to reach their customers. With the explosion in web traffic and numerous companies being highly dependent on the web for their operations, data mining of web related information (web mining) is becoming increasingly important. Web mining (Cooley et al. 1997, Cooley et al. 1999) allows companies to further understand their users' behaviour and demographic information, which in turn allows the organization to maximize sales (Eirinaki and Vazirgiannis 2003, Spiliopoulou 2000). It can also provide critical workload information, such as hits per user or session, enabling system administrators to improve usability, availability and reliability of their websites (Arlitt and Williamson 1997, Squillante et al. 1999).

The exploration in this area starts with Chapter 5. This chapter evaluates a technique for estimating reliability from server logs. The technique extracts

workload measures and error codes from these logs; reliability is then estimated based on the extracted information. Essentially, Chapter 5 is a "partial replication" of the original technique presented by Tian et al. (2004).

Through the study in Chapter 5, it is discovered that the session workload, which is the most popular unit, is often obtained using a static Session Timeout Threshold (STT) value ranging from 15 minutes to 2 hours. The values used do not consider the fact that many websites have different user profiles which means the STT will vary. Chapter 6 introduces a dynamic model that generates the STT for specific websites which allows the session workload to be estimated more accurately. This is important because having accurate data means better information can be mined. This allows organizations to improve quality attributes such as usability and functionality of their websites.

The research in Chapter 6 reveals that the distributional properties of the session workload unit are poorly understood. Whether the session workload can be described as a short-tailed or heavy-tailed distribution is a fundamental question for the investigation of the session workload unit. Hence, Chapter 7 empirically explores claims that the session workload can be described using a heavy-tailed distribution using many tests.

1.3 Contributions and Dissertation Outline

The outline of this dissertation, which includes discussions the contribution of each chapter, is as follows.

Chapter 2: Common properties contained in web application vulnerabilities are explored using a process similar to the GQM approach. The results show that web application vulnerabilities are primarily implementation vulnerabilities. They are caused through interactions between web applications and external systems. Furthermore, these vulnerabilities only contain explicit data flows, and are limited to relatively small sections of the source code.

Chapter 3: A white box approach is introduced to help eliminate web applications' vulnerabilities. This strategy allows investigators to accurately identify all inputs entering the web application and model the inputs as they reach external systems acting as data sinks. A case study using a commercial, currently deployed, mission-critical web application is presented to demonstrate the validity of the approach.

Chapter 4: An A-NIDS specifically for web applications called AIWAS is presented. The system attempts to learn the input specification associated with "normal" usage of the system, and validates any given input against this specification. A case study based on three web applications is performed to show the effectiveness of the system.

Chapter 5: The method of reliability estimation from server logs (Tian et al. 2004) is examined in detail. Two new websites are used with one having an extensive long data collection period. The error codes contained in the server logs are carefully explored to allow system administrators to focus on high value error codes. The workload models are re-examined to provide alternative methods for system administrators to analyze and interpret reliability information.

Chapter 6: A model, based on empirical observations, for estimating the Session Timeout Threshold (STT) is presented. Although the model has limitations, it provides an initial step that will allow future studies to expand upon. Furthermore, this model is proven to be applicable at many different resolutions and to two uniquely different websites. The concept that STT varies for each website is empirically proven. This encourages future research on web server logs to be performed using a customized STT value per website rather than a constant that's applied to all websites.

Chapter 7: The distributional properties of the session workload are re-examined. Additional tests such as Q-Q Plots and "wobble analysis" of the LLCD plots are performed to determine if session length can really be modeled by a heavy-tailed distribution. The results show that, for the samples used in the chapter, a method to accurately determine whether the session workload is drawn from a heavy-tailed distribution does not exist. Hence, the conclusion that they are drawn from such a distribution cannot be made.

Chapter 8: The conclusions and future works are presented in this chapter.

Chapter 2 – An Investigation into Web Applications' Vulnerabilities

Web applications have short release cycles and development time (Baskerville and Pries-Heje 2004). Many new features, enhancements and bug fixes are continually added during these cycles. Every change made to the system can introduce new security vulnerabilities. Using an approach similar to the Goal Question Metric approach (Basili et al. 1994), this chapter's goal is to help researchers improve the security posture of web applications by performing an empirical analysis of discovered vulnerabilities in 20 web applications to uncover any similarities in this sample.

Given the relative newness of the topic on web application vulnerabilities, limited factual or empirical information exists; hence, this chapter principally relies upon the researcher's previous experience with, and observations of, web applications. This has led to some tentative questions with regard to the vulnerabilities that exist within a wide cross-section of web applications; these questions are used to achieve the stated goal:

1. What proportion of security vulnerabilities in web applications can be considered as implementation vulnerabilities? The metric used to answer this question is the percentage of implementation vulnerabilities versus other types for the 20 applications under examination.
2. Are these vulnerabilities the result of interactions between web applications and external systems? The metric used to answer this question is the percentage of function calls to external systems that exist in the vulnerabilities.
3. What is the proportion of vulnerable LOC within a web application? That is, what is the vulnerability density? The metric used to answer this question is the number of vulnerable LOC versus the systems' total LOC.
4. Are implementation vulnerabilities caused by implicit or explicit data flows? The metric used to answer this question is the number of vulnerable code blocks (which are defined in Section 2.3.4) with implicit data flow and the number of variables assigned from an input.

Given the lack of solid causal theory utilized to derive the questions, it is believed that these questions should be viewed as an initial attempt in hypothesis formulation rather than an exercise in hypothesis confirmation or refutation. The remaining sections of this chapter are organized as follows. Section 2.1 introduces the terminology used in this chapter. Section 2.2 explains the survey and its procedure. Section 2.3 contains the metrics obtained for the four questions. Section 2.4 provides an overview of current techniques for detecting and eliminating web vulnerabilities.

2.1 Terminology

Several terms are defined in this chapter for the reader's convenience:

- **External Systems** – These are systems that the web application depends upon for its operation. For example, a shopping cart web application retrieves its product information from a Database Management System (DBMS), the external system.
- **EIV** – External Interaction Vulnerabilities. These vulnerabilities allow attackers to use vulnerable web applications as a vessel to transmit malicious code to an external system that can interact with the web application. The malicious code will modify the syntactic content of the information sent to the external application. In other words, EIVs allow attackers to target external systems that interact with the web application, rather than the actual web application itself

  Popular EIVs include SQL injections and cross-site scripting vulnerabilities. Any vulnerability is classified as an EIV if it has the following properties:
  - A malicious input is required to initiate the attack.
  - The malicious input is transmitted from the web application to an external system.
  - The malicious input does not exploit the web application directly. For example, buffer overflow vulnerabilities causing web applications to crash are not be classified as an EIV because they attack the applications' input buffers directly without interacting with an external system.
- **SQL Injection Vulnerabilities** (Scambray et al. 2006) – These vulnerabilities allow attackers to inject and execute SQL statements through the web application. For example, Figure 2.1 displays the pseudocode for a web application that asks the user for an email address stored in a database and displays the phone number associated with that email to the browser.

```
1.    $email = get_input();
2.    if ($email != RFC2822) {
3.      print "invalid email address";
4.      exit;
5.    }
6.    $sql = "SELECT phone FROM users WHERE email
                    =""+$email+""";
7.    $phone = query($sql);
8.    print $phone;
```

Figure 2.1 Example program

Statement 1 retrieves the email input address from the input. Statements 2-5 parses the input for a valid email address based on the RFC 2822[3],

[3] http://www.ietf.org/rfc/rfc2822.txt, last accessed July 25, 2009

which defines the standard format of an email address. Statement 6 builds a dynamic SQL statement based on the input retrieved. Statement 7 then instructs the DBMS to execute the SQL statement. Statement 8 prints the phone number retrieved from the email address entered. RFC 2822 allows many characters to be part of an email address which allow names with single quotes such as "O'Reilly" to be used in an email. Hence the user using a specially crafted address, which meets the specification, such as:

```
hi"' OR 1=1 --"@example.com
```

can embed a SQL statement. Using this email address, the expanded SQL statement becomes:

```
SELECT phone FROM users WHERE email ='hi"' OR 1=1 --
                    "@example.com'
```

Hence, this modified SQL statement is successfully injected.

- **Cross-site Scripting (XSS) Vulnerabilities** (Scambray et al. 2006) - These vulnerabilities allow an attacker to inject JavaScript/HTML code that other visitors to the website will execute. For example, an attacker can create a link to a vulnerable web application, such as

  http://www.site.com/?<script src=http://hacker.com/getcookie.js></script>

  which allows the attacker to become an administrator for that application.

- **Command Execution (Injection) Vulnerabilities** (Scambray et al. 2006) - These vulnerabilities allow an attacker to run various system commands ("`cd`", "`ls`", "`dir`", "`cat`", etc.) through the vulnerable system. An attacker, for example, exploiting this vulnerability can perform DoS (Denial of Service) attacks on the system by removing files essential to the application. Other system commands can be used to retrieve information or even alter the application's configuration settings.
- **Privilege Escalation Vulnerabilities** (Scambray et al. 2006) - These vulnerabilities allow an attacker to bypass the authentication system or escalate their privileges without using an injection attack. A typical vulnerable application would allow an attacker to access restricted sections without being identified as a valid user. For example, a web application can use a flag to identify administrators from normal users. This flag is stored in a hidden form field. The attacker, with knowledge of this flag, can manipulate it and escalate their account to gain additional (administrative) functions.
- **Information Disclosure (Leakage) Vulnerabilities** (Scambray et al. 2006) - These vulnerabilities allow an attacker, without using an injection attack, to access information not available to a normal user. Information disclosure differs from authentication bypass because authentication

bypass allows an attacker to perform tasks and retrieve information not available to them; whereas, information disclosure only allows the attacker to retrieve restricted information. For example, instead of displaying a generic error message when encountering an error, the web application can display the entire call stack which contains detailed information on the internal structure of the web application.

2.2 Survey

For this survey, 20 different applications implemented using six popular languages (PHP, ASP-VBscript, ASP.NET – C#, Java-JSP, Perl, and Python) were examined. The survey is explicitly limited to web applications; and hence several common languages (such as C) and vulnerability types (such as buffer overflows) are relatively uncommon within this domain.

2.2.1 Vulnerability Databases

Two popular vulnerability databases (VDB), the Open Source Vulnerability Database[4] (OSVDB) and the Bugtraq mailing list[5] were used to identify the vulnerabilities for these applications. These two databases provide information on known vulnerabilities for open source and proprietary products. Unfortunately, the survey requires detailed analysis of the source code, which is unavailable for proprietary systems; and hence the investigation is limited to open source systems. Although the complete survey for proprietary systems cannot be performed, the vulnerability types of 20 proprietary systems were briefly examined to determine whether they are similar to the vulnerability types found in open source systems. The results show that these proprietary systems have a similar distribution of vulnerability types.

Although the two databases have different maintainers, they are far from independent; in fact, Bugtraq can be viewed as a subset of OSVDB. OSVDB effectively collates information from all of the other major open-source vulnerability databases including: The National (U.S.) Vulnerability Database[6], US-CERT Vulnerability Notes[7]; Internet Security Systems - X-Force Database[8]; CERIAS Vulnerability Database[9], and the LWN security vulnerabilities database[10]. Hence, OSVDB can be considered as being a meta-source of information on this topic; and therefore, it is utilized as the basis of the selection procedure. Having said this, Bugtraq (due to its message board format) tends to include a more extended description of vulnerabilities than OSVDB, and hence this information source was always used, when it was available, to increase the understanding of the vulnerabilities.

---

[4] http://www.osvdb.org/, last accessed July 22, 2009
[5] http://www.securityfocus.com/archive/1, last accessed July 22, 2009
[6] http://nvd.nist.gov/statistics.cfm, last accessed July 31, 2009
[7] http://www.kb.cert.org/vuls/, last accessed July 31, 2009
[8] http://xforce.iss.net/, last accessed July 31, 2009
[9] http://www.cerias.purdue.edu/about/history/coast/projects/vdb.html,last accessed July 31, 2009
[10] http://lwn.net/Vulnerabilities/, last accessed July 31, 2009

2.2.2 Survey Procedure

The survey, for purposes of sampling, extracted vulnerability information covering the period between January 1, 2002 to May 31, 2007 from the OSVDB resulting in the records shown in Table 2.1.

Table 2.1 Number of vulnerabilities in the OSVDB

Total vulnerabilities	19,173
Products	5,175
Total web related vulnerabilities	7,290
Total web applications	2,695

OSVDB requires that all vulnerabilities be inspected to increase accuracy; unfortunately, Bugtraq has no such screening process. The survey worked with the vulnerabilities from OSVDB; the reliability of Bugtraq's vulnerability information was validated by comparing it with the corresponding entry from OSVDB. In addition, both databases encourage a product's developers to refute any vulnerabilities that they believe are incorrect, providing a further crosscheck of validity. None of the systems in this survey contained any disputed vulnerability.

The sampling procedure was to select randomly 20 open source web applications from the OSVDB database. However, these 20 web applications were required to meet certain criteria:

- They must have more than one update released.
- They must be larger than three KLOC.
- They must have vulnerabilities that are exploitable.
- They can be commercial systems, but the source code has to be available.

Table 2.1 shows that the selected web applications represent only a small fraction of the total number of web applications listed within the database. The results of the sampling process are shown in Table 2.2. Once the products were selected, the following steps were performed, on each product, to gather the necessary data for the analysis:

1. The source code for all applications was downloaded. This includes downloading older source code that contained the vulnerabilities of interest. This analysis requires the paths through the source code to be traced in detail. Hence, a requirement exists that effectively limits this type of survey to open-source type projects.
2. A source code counting tool (Practiline Source Code Line Counter[11]) was used to count the LOC for each application. Only files containing program statements were counted. The reported LOC does not include empty lines and comments.
3. Vulnerabilities for the applications were retrieved from the VDBs.

[11] http://sourcecount.com/, last accessed July 29, 2009

4. For each vulnerability, the source code was traced to the statements causing the actual vulnerability. Nested function calls are traced and stopped at calls to standard library functions.

Due to the different programming languages involved, different designs associated with each application and over 330 KLOC to examine, the entire process required about 1 year of effort. One week was required to study the OSVDB's relational diagram and to import OSVDB's data into a local database for faster access. One week was used to create a tool to query the database. Twelve weeks were used to study the programming languages. One week was used to install, configure, and deploy the web applications in a test environment. Ten weeks were used to study the web applications and the associated source code; four weeks were used to examine all the vulnerabilities associated with each application. Twenty six weeks were used to independently repeat the manual operations. This "verification" task was believed to be important as any manual task of this "length" is clearly error-prone and this approach is believed to have resolved any inconsistencies in the process.

2.2.3 Chosen Applications

Table 2.2 displays the examined applications and the number of vulnerabilities identified.

Table 2.2 Applications examined

Application	Description	Vulnerabilities	Language
A-CART	A commercial fully-featured shopping cart developed on the ASP platform using VBScript	8	ASP (VB)
AWStats	A popular open source log file analyzer for web/streaming/ftp/mail servers	5	Perl
Bonsai	An open source web-based querying front-end for CVS from the Mozilla Foundation	8	Perl
BugZilla[12]	An open source bug tracking system from the Mozilla Foundation	25	Perl
BugTracker.NET	A web-based bug tracker system that is currently used by thousands of development teams.	4	ASP.NET (C#)
Calcium	A commercial web calendar system by Brown Bear Software.	1	Perl
Daffodil CRM	A commercial open source customer relationship management system by Daffodil Software Ltd.	1	Java (JSP)

[12] Due to the numerous vulnerabilities reports available for BugZilla, the versions of the vulnerable systems are limited to 2.16.0 or higher.

DEV web management system	A content management system for web portals.	5	PHP
FileLister	A file system indexing tool	2	Java (JSP)
JSPWiki	An open source JSP-based WikiWiki engine	1	Java (JSP)
Mantis[13]	An open source tracking system	12	PHP
Neomail	A web-based email system; thousands of servers utilize the system.	1	Perl
PDF Directory	An open source software that generates a printable directory listing for any organization.	12	PHP
phpBB[14]	An open source popular message board system written in PHP that's being used on millions of websites.	23	PHP
ProjectApp	A commercial web-based project and task management system used for team communication by Iatek Corporation.	5	ASP (VB)
osCommerce	An open source e-commerce system, by osCommerce, currently being installed and utilized by 10,942 online stores.	15	PHP
Roundup	A full featured bug tracking system.	4	Python
sBlog	An open source blog system.	2	PHP
SkunkWeb	A robust, open source web application server.	2	Python
ViewVC	A browser interface for CVS and Subversion control repository.	2	Python
Total		**138**	

2.2.4 Tracing the Source Code

To determine the number of vulnerable LOC and how deep these statements are within the call stack, the source code for each known vulnerability was traced. Program slicing was first introduced by Weiser (1984) as a method of automatically decomposing applications. A slice of a program is a reduced, executable segment of the original program. A slice can be produced dynamically or statically. Static slicing techniques do not require input values whereas dynamic slicing techniques rely on some specific input to produce a slice (Tip

[13] Due to the numerous vulnerabilities reports available for Mantis, the versions of the vulnerable systems are limited to 1.0.0a1 or higher.

[14] Due to the numerous vulnerabilities reports available for phpBB, the versions of the vulnerable systems are limited to 2.0.7 or higher.

1995). Due to the lack of slicing tools for the languages examined, in this survey, a technique similar to dynamic slicing (Agrawal and Horgan 1990; Tip 1995) was used to produce contamination graphs (CGs) of the systems examined. The CG is not a SDG (system dependency graph), but rather a def-use graph that follows the malicious input from the entry point to the exit point of the system. While the technique used is similar to slicing, it does not produce complete slices of the system (hence, cannot be considered a slicing technique) and the graphs produced by the algorithm do not take into account object-oriented programming features such as inheritance and polymorphism; however, they contain sufficient information for this survey. More formally, a CG is a directed graph $G=<N,E_c,E_d>$, where N is a set of vertices corresponding to statements and control predicates, and E_c and E_d are the set of edges corresponding to the def-use data dependencies. The slicing criterion is C=(v, i, X*), where v is a variable in the system, i is an input value for v and X is a set of statements in the program. For this survey, v and i consist of variables and values that exploit the known vulnerabilities, while X* ($\subseteq$ X) consists of program statements where it is possible to export the vulnerability to an external system; and X is the entire set of statements in the program. The following algorithm is used to produce a CG for each v and i of interest.

1. DEF(w) is a definition of the variable w
2. USE(w) is a use of the variable w
3. Let V be a set of v
4. Let F be a set of statements; $F \subseteq X$; f_j be the statement at location j.
5. Let curloc be the program's current statement's location
6. Initialize V := {}; F := {}; prevloc := 0; prevDEFloc := 0;
7. Locate the first DEF(v) where v := malicious input
8. G := G + <curloc,{},{}>
9. prevloc := curloc
10. prevDEFloc := curloc
11. $V := v \cup V$
12. Execute program until $\exists\ v \in V \bullet$ USE(v)
13. If DEF(w) := USE(v) then
    a. G:= G + <curloc, prevloc→curloc, prevDEFloc→curloc>
    b. $V := w \cup V$
    c. prevDEFloc := curloc

    Else
    a. G := G + <curloc, prevloc→curloc,{}>
14. prevloc := curloc
15. If $f_{curloc} \in X^*$ then $F := f_{curloc} \cup F$
16. Go to 12 unless $F - X^* = \{\} \vee$ curloc = EOF $\vee$ program encounters an error due to a successful exploit.

An example of a CG using C = (keyword, "`<script>alert('hello')</script>`",{`query, echo, print`}) for an application examined, sBlog, is shown in Figure 2.2. The source code for this example is approximately 7,800 lines of PHP. Dotted directed edges on this graph represent DEF dependences (definition of a contaminated variable), while the solid edges represent USE dependences (usage of a contaminated variable). Each node is labeled with the source code's filename and the line where the statement can be found (in parenthesis). If a node represents a function call then it is labeled as "call 'function name'". System calls are also placed within the parenthesis. The graph above shows that the malicious input entered the system at line 36 of the search.php file. The solid edges show the transition between each USE statement. Nine lines of code use the malicious input (number of nodes) with five variables defined based on the malicious input (the number of dotted edges).

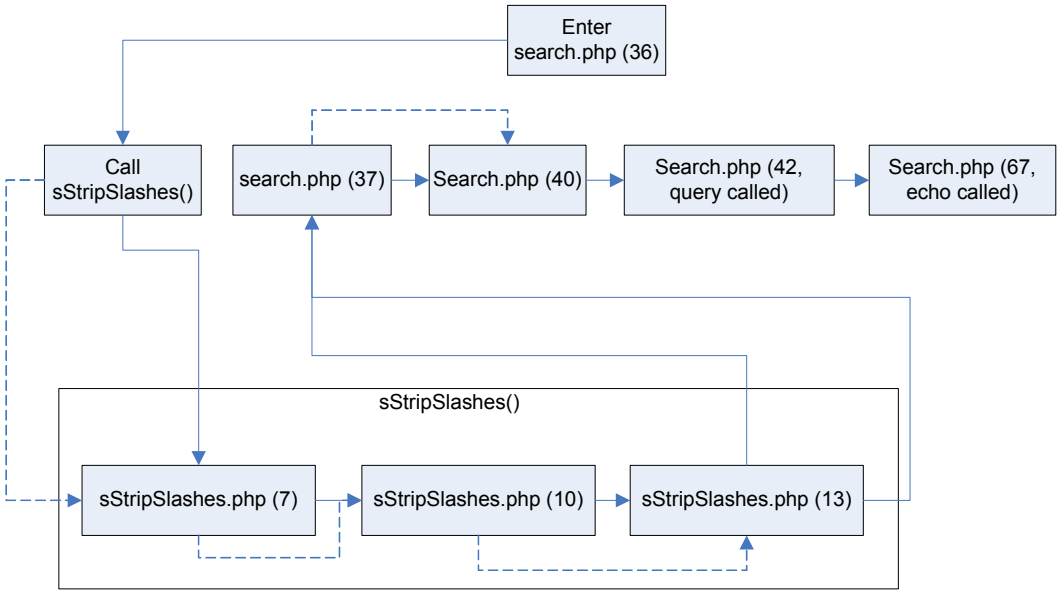


Figure 2.2 CG for sBlog

2.3 Results

This section contains the results from the survey. These results answer the four questions raised in the introduction and can be used to help the goal which is to improve the security posture of web applications by uncovering similarities between vulnerabilities.

2.3.1 Question 1

Question: What proportion of security vulnerabilities in web applications can be considered as implementation vulnerabilities?

Metric: The percentage of implementation vulnerabilities versus other types for the 20 applications under examination.

To answer Question 1, the known vulnerabilities are characterized into three categories based on Swidersky and Snyder's categorization (Swiderski and Snyder 2004):

- Architecture vulnerability: A vulnerability that is caused by a design flaw. For example, if the session ID generated by an application is easily guessable because the specification for a secure session management system does not have requirements on how IDs will be generated, such as a specific cryptographically hash routine, then the issue is considered architectural in nature.
- Implementation vulnerability: A vulnerability that is the result of an insecure coding practice. Using the same example as above, if the session ID is easily guessable because the cryptographically secure hash routine used to generate session IDs is written incorrectly then the issue is considered implementation in nature.

- Configuration vulnerability: A vulnerability that is caused by an incorrect configuration of the application; hence, if the vulnerability ceases to exist after an application is reconfigured, the vulnerability is classified as a configuration vulnerability. For example, the "register_globals" issue with PHP is considered a configuration vulnerability. This is a setting in the configuration file to instruct PHP to create global variables from the EGPCS (Environment, GET, POST, Cookie, Server) variables. When enabled, attackers can use the feature to define many global variables.

Table 2.3 shows the vulnerabilities and their distribution within the three categories defined. The standard error in the table is used to show the uncertainty of the value for each category. The equation for the standard error is:

$$\text{standard error} = \sqrt{\frac{p(1-p)}{n}} \quad (1)$$

where p is the probability of the sample belonging in a certain category and n is the sample size. This assumes that: n is small relative to the population size, the samples are selected from a simple random sampling process, and the sampling distribution of p is the binomial distribution[15]. Each category is treated independently from each other. For example, the first row of the table examines the implementation vulnerability. Hence, p is the probability of a vulnerability being an implementation vulnerability, and 1-p is the probability of it not being an implementation vulnerability.

This table answers Question 1 by showing that implementation vulnerabilities dominate; hence, addressing vulnerabilities within this category would allow a significant reduction in the number of vulnerabilities.

Table 2.3 Vulnerability category distribution

	number of vulnerabilities	% of vulnerabilities found in sample	standard error[16] (%)
Implementation	101	73.2	3.77
Architecture	30	21.7	3.51
Configuration	7	5.1	1.87

2.3.2 Question 2

Question: Are these vulnerabilities the result of interactions between web applications and external systems?

Metric: The percentage of function calls to external systems that exist in the vulnerabilities.

[15] Clearly, this is a simplification of the situation. However, the study has insufficient data to allow the evaluation of more complex models.

[16] In this context, the margin of error in the survey is approximately twice the standard error. Specifically, assuming a 95% confidence level, it is 1.96*the standard error.

Usually, these implementation vulnerabilities can be traced through a dynamic string, constructed from an input, being used in a function or method that allows the string to be passed to another system. The answer to Question 2 begins through the examination of the types of vulnerabilities within the implementation category. This examination reveals six different types of vulnerabilities are commonly discovered within web applications: SQL Injection, SQL Injection, XSS, Code Injection, Command Execution, Privilege Escalation, and Information Disclosure.

Table 2.4 displays the vulnerability types discovered during the survey. Close examination reveals that the majority of these types occur due to an interaction with an external system. These types of implementation vulnerabilities, bolded in Table 2.4, account for 95 of the 101 implementation vulnerabilities. While information disclosure may also be caused due to an interaction between the web application and the file system, this interaction is not obvious from Table 2.4, and the actual statements causing the vulnerability have to be examined to determine the exact cause.

Table 2.4 Implementation vulnerability types

	number of vulnerabilities	% vulnerabilities	standard error (%)
XSS	**56**	**55.4**	**4.23**
SQL Injection	**30**	**29.7**	**3.89**
Code Injection	**6**	**5.9**	**2.01**
Command Execution	**3**	**3.0**	**1.86**
Information Disclosure	5	5.0	1.45
Privilege Escalation	1	1.0	0.85

2.3.2.1 Vulnerability Types for Proprietary Systems

Since the vulnerability databases used also include proprietary systems, 20 of these systems were selected and examined to provide some level of comparison with the results found in the survey. Like their open source counterparts, these 20 applications were also randomly selected from the OSVDB. Table 2.5 shows the 20 applications examined.

Table 2.5 Proprietary systems

Application	Description	Vulnerabilities	Language
Active Auction House	A web based auction software designed for online auctions (ex. ubid.com, ebay.com).	7	ASP (VB)
AliveSites Forum	A component (COM) object tool that allow collaboration among members and users of a company or organization though the internet or intranet.	4	ASP[17]
ampleShop	A complete e-commerce system.	4	ColdFusion
AspDotNetStorefront	An ASP.NET shopping cart used by over 5,000 customers.	3	ASP.NET (C# and VB.NET)
ASPRunner	A web-based database management tool that provides administration for many popular databases.	7	ASP
Baseline CMS	A web-based content management system.	2	ASP
Bugzero	A web-based bug tracking, defect tracking, issue tracking, and change management system.	5	Java
Cisco CallManager Web Interface	The web-based interface for the Cisco Unified CallManager system.	3	ASP
couponZONE	A web-based system that provides online e-coupons.	2	ColdFusion
DUPortal Pro	An ASP-based Web Portal application.	11	ASP
E-School Management System	A web-based School Management Software	1	ASP.NET

[17] ASP and ASP.NET applications can be created using many programming languages. Due to the proprietary nature of the applications, the exact programming language used is unknown.

	designed to allow easy communication between students, teachers, parents & management.		
iCMS	A content management system.	2	ASP
Mall23 eCommerce	An e-commerce solution for Web Development and Hosting companies.	3	ASP
NetAuctionHelp	An ASP-based online auctioning system.	1	ASP
OneWorldStore	An e-commerce system that can be integrated to existing websites.	10	ASP
Revize CMS	A content management system.	5	Java
SCOOP!	Another web content management system for users without HTML knowledge.	7	ASP
SmarterMail	An advanced email and collaboration server.	5	ASP.NET
uStore	A dynamic storefront application for e-commerce websites.	3	ASP
Web Quiz	An easy application that for online test creations and assessments.	2	ASP

These 20 applications are commercial applications that either do not have their source code available or they require a developer's license to be purchased before the source code can be obtained. This table shows that ASP and ASP.NET is used for sixteen of the 20 web applications. Two out of 20 applications are powered by ColdFusion, which is the only scripting language that supports source code encryption without additional plug-ins or extensions. The remaining applications are created using Java technology.

Table 2.6 displays the vulnerabilities encountered for these 20 applications versus the vulnerabilities encountered for the 20 open source systems. This table shows that the top two vulnerabilities encountered on both types of system are XSS and SQL Injection, respectively. Code injection is less frequently encountered in

proprietary systems, which can be attributed to the fact that PHP remote file inclusion does not occur in these systems because these 20 systems do not use PHP. "Other" contains vulnerabilities that cannot be classified due to limited information provided for these vulnerabilities.

Table 2.6 Proprietary versus open source

	Proprietary		Open Source	
	% vulnerabilities	Standard Error (%)	% vulnerabilities	Standard Error (%)
XSS	48.8	4.26	55.4	4.23
SQL Injection	36.0	4.09	29.7	3.89
Code Injection	2.3	1.28	5.9	2.01
Command Execution	1.2	2.17	3.0	1.85
Information Disclosure	7.0	0.93	5.0	1.45
Privilege Escalation	1.2	0.93	1.0	0.85
Other	3.5	1.56	0	0

This table reveals that:

- The two types of systems agree that XSS and SQL injection (in that order) are the most numerous types of vulnerabilities experienced by web applications. Furthermore, the injection type vulnerabilities (SQL, XSS, code, command execution) combined to be the most popular vulnerability for web applications. This suggests that researchers interested in security problems associated with web applications should concentrate their efforts on these types of vulnerabilities. Clearly, this suggestion assumes that all vulnerabilities have a similar (negative) economic value.
- The two types of systems experience code injection problems at differing percentages. However, care needs to be exercised when considering this conclusion given the relatively low volume of these types of defects.

2.3.2.2 Mapping Vulnerabilities Down to Code Statements

Table 2.7 displays the statement types that cause the 95 EIVs. Several functions sharing the same properties are grouped into one family. For example, output statements such as `print`, `echo`, and `write` all send data to the browser, and hence they are grouped in the "`print`" family. Statements querying the DBMS such as `executeQuery`, `mysql_query`, `db.execute` are grouped in the "`query`" family.

Table 2.7. Statement usage

Statement Type	Number of Occurrences	Occurrence Percent (%)
"copy" file	**1**	**1**
dir	**1**	**1**
eval	**11**	**12**
file	**1**	**1**
open	**2**	**2**
preg_replace	**2**	**2**
"print" family	**47**	**49**
"query" family	**27**	**29**
require	**1**	**1**
system	**1**	**1**
wrong operator	1	1
Total	95	100

For every vulnerability, a CG was created using the technique discussed in Section 2.2.4. Statements resulting in the vulnerabilities can be located from these graphs. Table 2.7 highlights the statements used to call standard library functions which are the majority of the statements (99%). The statements listed in the table have the following behaviour:

- "`copy`" file - a function that allows programmers to copy an existing file.
- `dir` – a function that lists all files within a directory.
- `eval` – a function that accepts a string parameter and executes that string as a programming statement.
- `file` – a function that opens and reads a file based on a provided filename.
- `open` – a function that opens a file, pipe, or file descriptor.
- `preg_replace` – a function that will evaluate a provided string as a program statement if a special character is used (PHP only).
- "`print`" family – a group of functions that allows the application to send output to a browser.
- "`query`" family – a group of functions accepts a string containing one or more valid SQL statements and sends it to the underlying DBMS.
- `require` (PHP only) – a function that accepts a string parameter containing a filename (which contains programming statements), reads the file, then evaluates all the programming statements within that file. Similar functions include include and include_once (PHP only).
- `system` – a function that accepts a string parameter containing a system's command, then creates a new process and executes the command.

- typographical error – this is a statement where the programmer used the wrong operator for a conditional branch. For example, instead of using the < operator in an `if` statement the programmer used the <= operator. This operator does not enable an interaction and is the exception to the general rule.

Based on Table 2.7, the implementation vulnerabilities can be divided into two categories:

1. Interaction with external systems (EIV – External Interaction Vulnerability).
2. Wrong statement usage.

Table 2.7 shows that 99 percent of the implementation vulnerabilities are EIVs; this answers Q2. This answer means developers should concentrate on the data flow between the web application and other systems because this is where most of the vulnerabilities occur.

2.3.3 Question 3

Question: What is the proportion of vulnerable LOC within a web application? That is, what is the vulnerability density?

Metric: The number of vulnerable LOC versus the systems' total LOC.

Alhazmi et al. (2007) have explored the vulnerability density for Operating Systems and discovered that the density is very low. In this study, the vulnerability density for web applications is explored. It is believed implementation vulnerabilities are also limited to relatively small portions of the entire web application. That is, the number of vulnerable LOC is significantly smaller than the total LOC of a web application.

To answer Question 3, each implementation vulnerability was traced using the method outlined in Section 2.2.4. A total of 101 graphs for the 20 applications were generated. To determine the complexity of the vulnerable code, the number of nodes per graph and contaminated variables per graph were examined. Figures 2.3 and 2.4 show that the majority of the graphs have less than five nodes and four contaminated variables. In fact, 70% of the CGs contain less than five nodes and 93% of the CGs contain three or less contaminated variables. Hence, the majority of the vulnerabilities can be viewed as "small and manageable". In fact, even the largest number of statements and contaminated variables associated with a vulnerability (15 and 12 respectively) is quite small when compared to the overall size of the system.

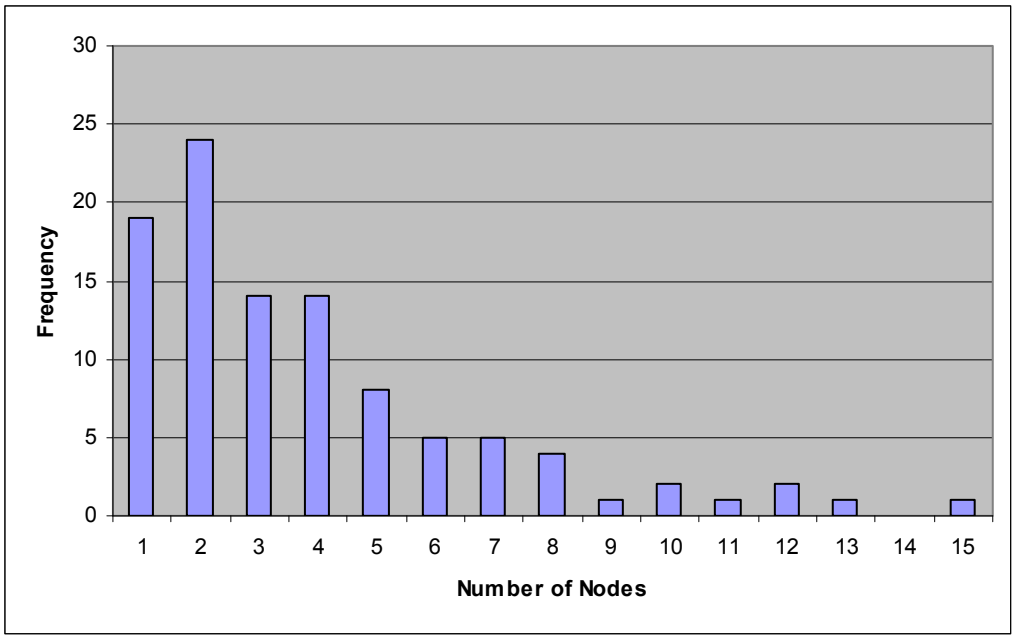


Figure 2.3 Histogram of nodes showing many CDGs have less than 5 nodes

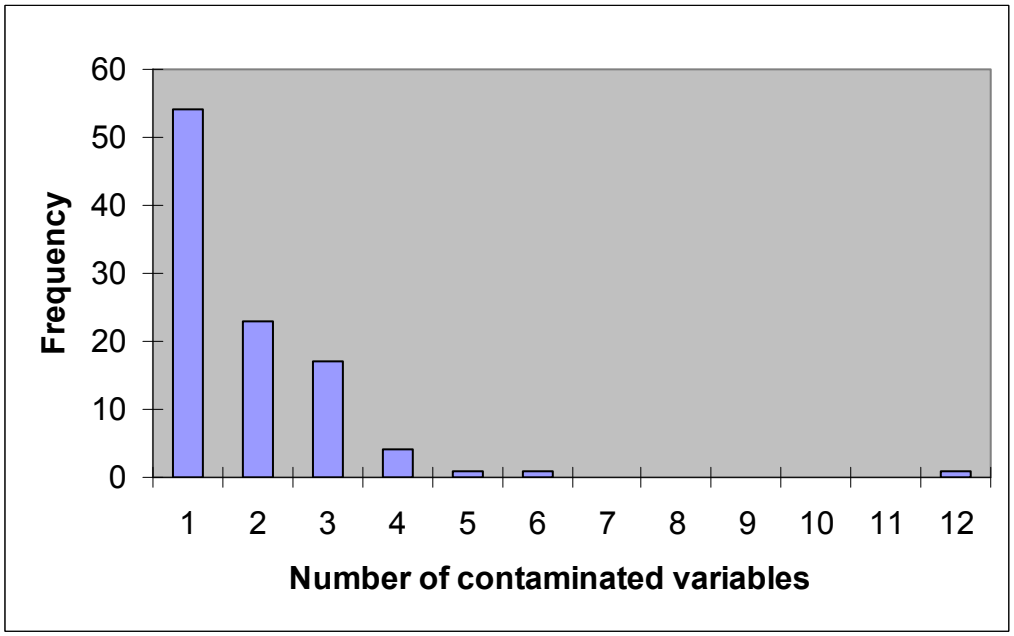


Figure 2.4 Histogram of contaminated variables showing many CGs have less than 4 contaminated variables

Once the CGs are obtained, the vulnerable LOC contained within each CG was counted. Table 2.8 further demonstrates that the number of vulnerable LOC for the known vulnerabilities is significantly smaller than the overall LOC. The results from Figures 2.3-2.4 and Table 2.4 provide the answer to Question 3 which is that vulnerability density is small. Since Figures 2.3 and 2.4 and Table 2.8 show that the number of vulnerable LOC is small compared to the overall size

of the system, it can be beneficial to introduce a solution to solve implementation vulnerabilities by concentrating on the CGs with vulnerable LOC.

Table 2.8 Vulnerable LOC versus Total LOC

Application	Total LOC	Vulnerabilities	Vulnerable LOC
A-CART	4,067	8	8
AWStats	26,688	5	9
Bonsai	6,980	8	42
BugTracker.NET	18,101	4	4
BugZilla	9,306	4	39
Calcium	39,348	1	2
Daffodil CRM	25,221	1	4
DEV web management system	11,434	5	5
FileLister	9,139	2	12
JSPWiki	21,231	1	4
Mantis	25,295	12	50
Neomail	1,438	1	5
osCommerce	38,833	15	34
PDF Directory	9,451	12	38
phpBB	29,812	23	100
ProjectApp	11,444	5	11
Roundup	27,061	4	8
sBlog	7,844	2	12
SkunkWeb	6,554	2	4
ViewVC	7,549	2	2

2.3.4 Question 4

Question: Are implementation vulnerabilities caused by implicit or explicit data flows?

Metric: The number of vulnerable code blocks with implicit data flow and the number of variables assigned from an input.

Implicit data flows are information flows via the control structure of the program (Denning and Denning 1977). For example, the statement "`if (y == true) then x:='a'; else x:='b'`" shows that variable `y` implicitly defines the value of variable `x`. Hence, there is an implicit data flow from variable `y` to variable `x`. To obtain implicit flow information, conditional branching statements for all the nodes from the CGs generated in Section 2.3.3 were manually examined. The examination revealed 56 statements with conditional branching from the 101 CGs. The code blocks for each of these statements were inspected for any implicit data flows. A code block is defined as a block of code that is part of the conditional branch. For example, the following conditional statement

would contain two code blocks with the first code block containing an implicit data flow:

```
if (x=1)
  y := 2;
  print y;
else
  call func(x);
end
```

The above example shows that if a CG has statements like those in the first code block, the CG would contain an implicit data flow. Table 2.9 shows the results of the code block investigation. The 56 statements with conditional branching lead to 83 code blocks. Twenty-nine of these code blocks do have implicit data flow. However, none of these code blocks with implicit data flows are part of the CGs obtained in Section 2.3.3.

Table 2.9 Code blocks

Number of CGs containing conditional statements	56
Number of code blocks inspected	83
Number of code blocks with implicit data flows	29
Number of CGs containing code blocks with implicit data flows	0

Table 2.9 shows that the CGs do not contain any implicit data flow statements. To determine if the code blocks containing implicit data flows can lead to potential vulnerabilities, a further investigation on the 29 code blocks was performed. Thirty-six variable assignments were discovered in these code blocks. The variable assignments are either from constants or pre-existing variables. The two example code blocks below shows two possible methods for the variables to be assigned. The first code block shows that the variable is assigned from a constant. The second code block shows the variable being assigned from an existing variable.

```
if (isset($_GET['admin'])
  $admin_mode = 1;
end
```

```
if (strlen($_POST['msg']) < 20)
  $error = $too_short;
  $print($error);
end
```

Although it is clear that constants are generally safe from implementation vulnerabilities, the pre-existing variables need to be examined to determine the original source of the data. A back-trace for each variable assigned from an existing variable was performed; if the variable can be traced to an input, then the potential for vulnerabilities exists. The results can be seen in Table 2.10.

Table 2.10 Variable being assigned from different sources

Number of variables being assigned from a constant	9
Number of variables being assigned from an existing variable	27
Number of existing variables initialized from a constant	27
Number of existing variables initialized from an input	0

The results from Tables 2.9 and 2.10 provide an answer to Question 4. That is, implicit data flows do not lead to any vulnerabilities in the systems examined. Hence, without further evidence, efforts on eliminating implementation vulnerabilities can focus on explicit data flows.

2.4 Background

Although studies on web application vulnerabilities properties currently do not exist, many techniques and approaches to detect, or mitigate against, web vulnerabilities have been proposed. In this section, these techniques are briefly presented and discussed.

SQLrand (Boyd and Keromytis 2004), AMNESIA (Halfond and Orso 2005), SQL-Guard (Buehrer et al. 2005), SQLCheck (Su and Wassermann 2006), CSSE (Pietraszek and Berghe 2005), WASP (Halfond et al. 2006, 2008), SQLProb (Liu et al. 2009) are all approaches aimed at addressing SQL injection vulnerabilities. SQLRand inserts random tokens into SQL statements and uses a proxy server to translate these tokens. An incorrect query can be detected if the SQL query does not contain the correct tokens. AMNESIA, SQLGuard and SQLCheck are all model-based approaches. AMNESIA uses static analysis and runtime monitoring to detect for SQL injection vulnerabilities. Static analysis is used to build models of the SQL statements, while the runtime engine detects whether the query strings matches the models. SQLGuard requires the developers to call special functions to build a model of the SQL query to be used. SQLCheck uses a formal definition of a SQL injection vulnerability; and identifies SQL injection attacks based on the formal definition. CSSE and WASP are dynamic approaches designed to address SQL injection vulnerabilities using taint analysis. These approaches attempt to mark negative tainting (CSSE) or positive tainting (WASP) to identify malicious query statements, before they are passed onto the DBMS. Both approaches involve modification to either the runtime engine or usage of a specialized API. SQLProb uses a proxy to identify SQL injection attacks before they reach the web application.

Other approaches to applications' security have also been proposed which address all types of web application vulnerabilities. Security Gateway proposed by Scott and Sharp (2002) is an application firewall that filters out all malicious inputs before they reach the web application. Nguyen-Tuong et al. (2005) proposed a dynamic approach to detect attacks through taint analysis. Martin et al. (2005) proposes PQL (Program Query Language) that enables programmers to specify a sequence of events between objects. Balzarotti et al. (2007) presents a static analysis approach capable of detecting both workflow attacks and data-flow

attacks. WebSSARI (2004) combines static analysis with a runtime component to check on the static model. Pixy (2006) is currently one of the more advanced static taint analysis tools available for PHP. Shankar et al. (2001) proposed a static approach that can detect format-string vulnerabilities commonly found in C-based applications. The method defines two extended data types, tainted and untainted, which help reduce the amount of false positives generally associated with static analysis methods. Zhang et al. (2002) and Johnson and Wagner (2004) further extend the approach by using it to assess security issues with the Linux Security Modules framework and user/kernel pointers successfully. These approaches are designed to detect vulnerabilities in C-based applications, and hence, their effectiveness with scripting languages used to develop web applications such as PHP, Ruby, and Python remain unknown.

Scanning tools also exist to help developers and system administrators identify vulnerabilities. QED (Martin and Lam 2008) and Ardilla (Kiezun et al. 2009) attempt to generate SQL Injection and XSS attacks automatically. Secubat (Kals et al. 2006) and other commercial web scanners, such as Acunetix Web Vulnerability[18] Scanner, extend bypass testing by creating tools that provide automatic penetration testing for web applications without using the web applications' target clients. Lin and Chen (2006) extend traditional black-box testing techniques with elements of static analysis by including a tool to automatically inject guards at input points found through the crawling component.

[18] http://www.acunetix.com/, last accessed February 7, 2010

Chapter 3 – Practical Elimination of External Interaction Vulnerabilities in Web Applications

Many approaches designed to address External Interaction Vulnerabilities (EIVs) have been proposed – these approaches are discussed in Section 3.5 – further confirming that EIVs are an extremely important class of vulnerabilities for web applications. Current approaches are either: application security (McGraw 2004) oriented, static analysis methods or black-box techniques. White-box approaches to detecting all EIVs are not common in the research literature nor in industrial settings. In this chapter, a practical white-box software development process that can help detect and eliminate web applications' EIVs is introduced. The approach builds a model based on the data flow of the application. The approach is significantly enhanced by computer-support software which automates much of the "straightforward" components in the approach, allowing the security team to concentrate of the "creative" components in vulnerability detection. This partial automation strategy also makes the approach highly effective in terms of effort and maximizing the quantity of vulnerabilities discovered. The partial automation strategy utilizes two pre-existing tools (a crawler and a capture replay tool) and two purpose-built proof-of-concept tools, Web Application Input Collection (WAIC) and Web Application Graph Generation (WAGG), to automate portions of the process for the web application in the case study. The strategy can be combined with previous approaches to further harden web applications against EIV related attacks.

The remaining sections of this chapter are organized as follows: Section 3.1 defines EIVs. Section 3.2 provides an overview of the research problem. Section 3.3 introduces EIV analysis. Section 3.4 presents an industrial case study for the presented strategy. Section 3.5 provides an overview of current approaches aimed at addressing EIVs.

3.1 Definition

External Interaction Vulnerabilities (EIVs) are vulnerabilities that allow attackers to use vulnerable web applications as a vessel to transmit malicious code to an external system that can interact with the web application. The malicious code will modify the syntactic content of the information sent to the external application. In other words, EIVs allow attackers to send systems commands that interact with a web application, rather than the actual web application itself. Currently, four interaction categories are defined:

- DBMS interaction – this is the interaction between the web application and external DBMS. An example of a DBMS interaction would be a web application calling a "query" function to send a SQL statement to a DBMS.
- Browser interaction – interactions in this category are between web applications and clients (typically a browser). An example of a browser

interaction would be a web application sending an HTML encoded webpage to a web browser.

- OS/Filesystem interaction – this is the interaction between the web application and the filesytem or operating system. An example of this interaction type would be a web application reading a configuration file from the hard drive.
- Interpreter interaction – interactions in this category are between the web application and a programming language interpreter (usually the same language as the web application). An example of an interpreter interaction would be a web application calling "eval" to execute a programming statement.

Popular EIVs include SQL injections and cross site scripting vulnerabilities. A vulnerability is classified as an EIV if it has the following properties:

- A malicious input is required to initiate the attack.
- The malicious input is transmitted from the web application to an external system.
- The malicious input does not exploit the web application directly. For example, a buffer overflow vulnerability cannot be classified as an EIV because it attacks the application's input buffer directly without interacting with an external system.

3.2 Research Problem

As with many research problems, a precise specification of the problem of interest is difficult to comprehensively frame, and is only likely to be available after the problem has been completely solved. However, this research does not seek to address all aspects of vulnerabilities; rather it is a specific problem which is framed with the following constraints or objectives.

- The work is only interested in web applications and EIVs. However, any solution should be applicable to all types of web applications and seek to eliminate all types of EIVs. As web applications become increasingly reliant on other external systems, such as other web services (Curbers et al. 2002, Alonso et al. 2003) or NXDs (Chaudhri et al. 2003), new types of EIVs will emerge. For example, XPATH[19] is becoming increasingly popular technique for querying XML documents. A web application that uses XPATH can be vulnerable to XPATH injection, which is a type of EIV. Although the number of exploits based on XPATH injection vulnerabilities is currently small compared to XSS and SQL injection, this number will only increase as more and more web applications take advantage of XPATH as a method of retrieving data from XML documents. A solution that cannot support future or, currently obscure, EIV types will quickly become obsolete. Web application technology moves at an incredible pace. Within a few years, web applications have evolved from simple guestbooks and web counters which relied on flat-text files for data support to fully interactive office productivity suites that

[19] http://www.w3.org/TR/xpath, last accessed February 9, 2010

interact with enterprise third party systems such as Oracle DB. A solution that cannot keep pace with the evolving web applications would not be practical for industrial use.

- Any solution must support a wide range, including multiple versions, of external systems. This can be viewed as a large configuration problem – see Eaton and Memon (2007) for work in this area. Web applications can interact with many different external systems. For example, one application may interact with Internet Explorer 6.5 and MySQL 3.23; another application may interact with Internet Explorer 5.5, Mozilla FireFox 1.5, SQLite 3.4.2 and Google Maps API 2.1. While similar, different versions of external systems will have different interfaces. These differences often cause highly vulnerable situations as systems commonly fail to correctly adapt to these evolving interfaces. For example, only Internet Explorer 6 SP1 and later support HTTP-Only cookies[20]. This is an extension to the Set-Cookie header that mitigates XSS attacks targeting information stored within cookies. However, not all IE versions support this extension, and hence, some IE versions have a much higher risk of being vulnerable to XSS attacks targeting cookies than other IE versions. Furthermore, Mozilla FireFox 2.0.0.4 and lower only support HTTP-Only through an extension. Hence, the same version of FireFox (for example, 2.0.0.4) can have different risk levels, with regard to XSS attacks targeting cookies, depending on whether the HTTP-only cookie extension is enabled.
- Any solution must be language-independent. This is important as web applications utilize a wide range of scripting/programming languages (Java, Visual Basic, PHP, Perl, C#, Python, JavaScript, Ruby, Cold Fusion, etc.), which support a variety of different programming paradigms and styles. Furthermore, many web applications, such as AJAX enabled applications, utilize more than one scripting/programming language. Hence, any solution that can only support a single scripting/programming language would not be usable against these multi-language applications. This objective becomes more important as AJAX enabled web applications, such as Google Docs & Spreadsheets, and Mashups[21], which combines multiple web APIs in one hybrid web application, become more popular.
- Any solution must be applicable to current industrial strength web applications. Apart from the constraints stated above, this constraint is not too demanding. Current web applications are relatively small in scale (the previous chapter shows that a large number of these systems, range in size from 4 to 40 KLOC); and hence, many of the restrictions placed by ultra-large scale systems are not of great concern here.
- Any solution must be "practical" in an industrial sense. Industrialists often express their frustration that many exciting pieces of software research are

[20] http://msdn2.microsoft.com/en-us/library/ms533046.aspx, last accessed February 9, 2010

[21] http://www.ibm.com/developerworks/xml/library/x-mashups.html, last accessed February 9, 2010

not applicable in their context. Research which requires large-scale retraining or complete redefinitions of their life-cycles are often considered by industrialists as "impractical". Hence, this research only seeks solutions which can be viewed as an incremental development of most life-cycles, and solutions which can be utilized by many practitioners with minimal additional training or with on the job training.

In summary, the research problem can be viewed as a two-level problem. The lower-level problem is to find all EIVs that exist within a web application. The higher-level component, which generalizes the lower-level problem to cover all web applications, can be viewed as a large configuration space (CS): $L \times ET \times NE$, where L is the set of scripting/programming languages used to build web applications, ET is the set of EIV Types, and NE is the set of EIVs. Furthermore, NE is defined as $ES \times VES$ where ES is the set of different external systems and VES is the set of versions of these external systems.

3.3 External Interaction Vulnerability Analysis

The proposed strategy can be thought of as a white-box technique (Myers 1979); EIV analysis is performed using the following steps; these steps are further discussed in Sections 3.3.1 – 3.3.4:

1. Create a sitemap for the web application.
2. Identify all input sources.
3. Create contamination data graphs (CDG).
4. Test the contamination flow graphs until a coverage criterion is met.

Although the CDG generation step of EIV analysis is similar to static analysis, the two approaches are not the same. CDG generation is just one of the four steps required for EIV analysis. With EIV analysis, CDGs are a resource that security practitioners can utilize to uncover EIVs, whereas static analysis would simply present the CDGs without any further instructions on how these results should be handled. Section 3.3.3 discusses the difference between CDGs and DEF/USE approaches used in traditional data flow analysis approaches. Hence, EIV analysis is most appropriately classified as specialized data flow testing that concentrates on tainted data flows. However, this approach is not a dynamic taint analysis approach as proposed by other researchers (Halfond et al. 2006, Pietraszek and Berghe 2005, Xe et al. 2005). It does not contain a runtime component that monitors the application's memory for tainted values. In fact, dynamic taint analysis approaches often require modifications to the runtime platform or extra software which can complicate configuration and reduce performance.

3.3.1 Creating the Sitemap

The sitemap is a critical part of EIV analysis because it allows the security practitioner to identify path executions for EIV analysis. A sitemap is a set of directed graphs that represents a model of all accessible web pages from a web application. Each of these graphs contains a set of edges and nodes. More formally the sitemap is defined as $S = \{G\}$ where:

- G = <N, E> where
    - N is a set of nodes representing the web pages. Each $n \in N$ represents a web page accessible by the client.
    - E is a set of directed edges. Each $e \in E$ from node n_1 to n_2 shows that n_2 is reachable from n_1.

Although the sitemap can be generated manually, the process is labour intensive and not very practical; hence a web crawler (Heydon and Najork 1999, Moody and Palomino 2003) is used to speed up the process. Specialized crawlers that can handle dynamic contents (Raghavan and Garcia-Molina 2001), or site specific pages (Miller and Bharat 1998) also exist. A practitioner can use a crawler that can handle dynamic content to help create the site map for the web application under investigation; however, because crawlers can only follow web pages through links or forms, the practitioner's intervention is required in order to generate a complete sitemap. However, crawlers can only access web pages that are referenced from other pages; hence, if a web page is "hidden", no other web pages link or refer to it, then it cannot be accessed by the crawler. In order to create complete sitemaps, the number of web pages crawled needs to equal the total number of web pages for the web application. For example, consider a web application that has eight web pages: index, $normal_2$…$normal_5$, indexadmin, $admin_2$ and $admin_3$. These eight web pages comprise two distinct sections:

- One section is accessible to normal users. This section contains five web pages called index, $normal_2$…$normal_5$.
- Administrators can only access another section. This section contains 3 web pages called indexadmin, $admin_2$, and $admin_3$.

These two sections are separated and do not cross-reference each other; hence, the crawler needs to be executed twice.

The crawler must be configured to exclude pages not belonging to the web application, usually by restricting the crawling operation to a single domain or a directory of a web site. The crawling operation should not be limited to a single IP if the web application is hosted on multiple servers because the IP addresses for these servers are different. Although the IP addresses for these servers are different, the domain remains the same. For example, Amazon.com uses several servers to power its e-commerce application, but all the servers are under the Amazon.com domain.

3.3.2 Inputs

Inputs for web applications come from many sources (Wheeler 2003) including clients (browsers). Black-box techniques for web applications primarily concentrate on this source of input (Offut et al. 2004, Tappenden et al. 2005); however, investigating this source alone is insufficient. Many web applications communicate with external applications to perform required tasks. Inputs from these external applications cannot be trusted and need to be examined to reveal all possible security faults. For example, Figure 3.1 shows a sequence diagram of a simplified interaction between a client and a search engine. The client sends a request to the search engine web application; the search engine then parses this

request, creates an SQL statement and sends it to a DBMS. Once the results are retrieved from the DBMS, the search engine builds an HTML page and returns this page to the client. This interaction sequence has two input sources, one from the client (the search query that the user sends) and the other from the DBMS (the results that the DBMS returns). Code segments using these inputs have potential vulnerabilities associated with them. If the search engine does not parse the input from the DBMS then an attacker may compromise the DBMS and insert a JavaScript payload, which the search engine will return to the client after a search request. In this scenario, the search engine is vulnerable to a stored XSS[22] attack. Hence, if the security practitioner only examines the input from the client, the stored XSS vulnerability from the second input source will not be revealed until the product is released.

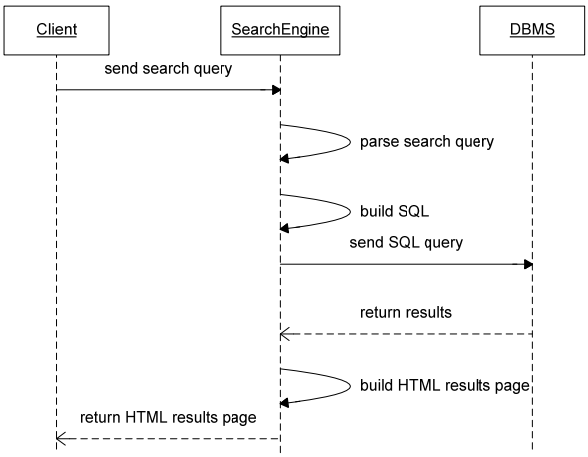


Figure. 3.1 A search sequence

3.3.2.1 Input Classification

Ideally, all inputs should be examined; however, software development companies have time and budget constraints limiting the amount of testing. To aid with the selection of inputs to be investigated, inputs are classified into two types:

- Inter-organization inputs – these are input values from unknown sources.
- Intra-organization inputs – These are input values entered by known and believed to be trusted sources (administrators, webmasters, employees, etc.). For example, a news article entered into a CMS (Content Management System) by an editor is considered as an Intra-organization input; whereas a comment posted by an anonymous user to a news item is considered an inter-organization input.

Inter-organization inputs should have a higher priority because, on average, they represent greater risks to the system. Intra-organizational inputs should still be examined because attacks can still happen under specific circumstances. For example, a spiteful employee can intentionally attack the system, or an attacker

[22] http://www.owasp.org/index.php/Cross_Site_Scripting, last accessed January 20, 2008

can access an employee's username and password through phishing (Dhamija and Tygar 2006, Ollman 2004) or other social engineering techniques (Granger 2003), and use the account as a mechanism to inject payloads.

Multiple inputs from the same source do not imply that they are of the same type. For example, consider a simple e-commerce system that can display a product's name, price, and user reviews. All three data are retrieved from three columns within a DBMS. However, an employee enters the product's name and price, while web visitors, who claim to have used the product, enter the user reviews. In this scenario, although the inputs are from the same source (DBMS), the two columns containing the values entered by the employee (product's name and price) are considered as intra-organization inputs while the other column containing the user reviews are considered as inter-organization inputs. Under constraints, a security practitioner can prioritize and examine the e-commerce system's ability to verify and validate the inter-organization inputs (reviews submitted by users) first, before investigating the intra-organization inputs (product's name and price).

3.3.2.2 Input Identification

To identify all inputs, the security practitioner will need to have access to the source code. Each input that enters the system can be stored in multiple variables; these variables are the starting nodes for the contamination graph. To allow automation, a formal model for the inputs is created.

Each source code file can have zero or more inputs. An input unit (IU) = (S, T, N) is used to describe inputs where:

1. S = The source of the input. This specifies which external system supplies the input value.
2. T = The type of the input. $T \in \{\text{Inter-organization, Intra-organization}\}$.
3. N = an ordered pair (v, l) where v is a variable that stores the input value and l is the location where the variable is defined. In other words, $(v,l) = N \text{ iff } (DEF(v) := \text{input value} \wedge LOC(v) = l)$ where DEF(v) is the statement that defines v and LOC(v) is the location where v can be located which is the line number and the filename.

To introduce the algorithm used to identify the inputs, several variables and functions need to be defined:

- Let I be a set of IUs.
- Let F be a set of source code files.
- Readlines(f) is a function that returns a set of statements in file f.
- Source(v) is a function that returns the source of variable v. That is, it returns the external system that sends a value to the application under test.
- Type(v) is a function that returns the type of variable v.

The following algorithm derives all inputs for a web application. The algorithm requires the set of source code files to be known. The inputs generated from the algorithm are stored in I:

```
1.  I = {};
2.   ∀(f ∈ F) {
3.     ST := Readlines(f);
4.     ∀ (st ∈ ST) {
5.        if (st = (DEF(a) := input)) {
6.           I := I ∪ (Source(input),Type(input),(a, LOC(a)));
7.        }
8.     }
9.  }
```

The algorithm parses all source code files to search for statements where variables are initialized from an input.

3.3.3 Contamination Data Graphs

Contamination data graphs (CDGs) are a critical component of the EIV analysis process. These graphs will allow the tester to design test cases that can reveal potential EIVs for the web application under investigation. CDGs are a variation on DEF/USE graphs used in data flow testing (Frankl and Weyuker 1998, Harrold and Rothemel 1994, Laski and Korel 1983, Rapps and Weyuker 1985). Liu et al. (2000) has extended the technique for web applications; however, the approach concentrates on inter-procesure and intra-procedure data flows and not intersystem data flows which are critical to the EIV analysis process. Hence, this study introduces an intersystem contamination graph (CDG), which describes the path an input value travels upon entering the system under investigation to reach various external systems. The graph is similar to the taint variable concept (Hurst 2004).

A CDG differs from a traditional DEF/USE graph because it does not contain all statements within a program. Its nodes only contain DEF/USE statements related to the input that initializes the graph. The CDG's purpose is to trace the path of an input value from its entry point to its various exit points (i.e. statements that send the input value to external systems). A CDG graph is formally defined as CDG = < N, E, N_e> where:

- N is a set of nodes representing all statements containing either a DEF or USE instruction.
- E is a set of directed edges representing the data flow between statements. Each e ∈ E, from nodes n_1 to n_2, shows that the flow of the tainted data moves from n_1 to n_2.
- $N_e \subseteq N$ is a set of exit nodes where the input values exit the system and are transmitted to external systems.

The security practitioner needs to create a CDG graph for each input identified in Section 3.3.2. Before the algorithm used to create the CDG is introduced, several variables and functions need to be defined:

- V is a set of variable names.

- getLoc(x) returns the location (loc) of the variable defined in the input unit x. loc is comprised of a line number and a filename. Section 4.B discussed the model for the input.
- getVariable(x) returns the variable name (v) of the variable defined in the input x.
- getStatement(loc) returns the statement at location loc.
- getNextUse(V,loc) returns the location of the next statement that contains a USE instruction for one of the variables in the set V starting from the location loc. If a statement cannot be found before the end of the program is reached, then getNextUse(V,loc) returns EOP (End of Program) stating that no additional statements can be found.
- getPrevUse(V,loc) returns the location of the previous statement containing a USE instruction for one of the variables in the set V from the location loc. If the current location is the first USE within a branch, then it returns the last encountered USE instruction before the branch.
- EXITPOINT is the statement that sends the value of the variable stored in the input unit to an external system. For example, a system call to the print function is an exit point if the value of the variable stored in the input unit is passed into the print function.

The following algorithm can be used to produce a CDG for each input unit:

```
1.  create_CDG(inputUnit) {
2.      N := {};
3.      E := {};
4.      Ne := {};
5.      V := {}
6.      loc := getLoc(inputUnit);
7.      var := getVariable(inputUnit);
8.      V := var ∪ V;
9.      N := loc ∪ N;
10.     loc := getNextUse(V,loc);
11.     if (loc != EOP) {
12.         N := loc ∪ N;
13.         E := (getPrevUse(V,loc)→loc) ∪ E;
14.         st := getStatement(loc);
15.         if (st = DEF(w) {
16.             V := w ∪ V;
17.         }
18.         if (st = EXITPOINT) {
19.             Ne := loc ∪ Ne;
20.         }
21.     } else {
22.         Go to Step 6;
23.     }
24.     return <N,E,Ne>;
25. }
```

The algorithm starts at the location where the input enters the system. It then searches for all statements utilizing the value and all variables assigned with the value. Finally, all exit points are then identified and flagged accordingly. Hence, using the above algorithm, a complete data flow path, from the entrance to the exit points, for the input is created.

The algorithm has a well-known limitation – it is unable to follow information through implicit flows (Denning and Denning 1977) if constants are used to initialize the variable in the flow. For example, *if (INPUT == 1) then x:='a'; else x:='y'*. However, an approach which adequately resolves this limitation is unsolvable. Further, while implicit flows can exist in any program, their frequency of occurrence is not well known. In addition, the previous chapter found that none of the paths through the compromised systems which lead to vulnerabilities contained implicit flows. Hence, while this theoretical limitation exists in this testing process, there exists no compelling empirical argument that the limitation causes the process to be inadequate on a regular basis when applied to realistic systems.

An Example of the CDG

In this section, an example program is used to demonstrate the creation of a CDG. The following is a highly simplified search application:

```
1. var search_keyword = gets();
2. var sql_query = "SELECT text FROM contents WHERE text like
   '%"+search_keyword+"'";
3. var results = execute_query(sql_query);
4. if (results != EMPTY) {
5.    print "Your keyword: "+search_keyword+" returned the following
   results:";
6.    print results;
7. } else {
8.    print "Your keyword: "+search_keyword+" returned no results.";
9. }
```

This application contains two input sources, one from the user (line 1) and one from a DBMS (line 3); hence two CDGs are required to be generated. Figures 3.2 and 3.3 show the CDGs created for these two inputs.

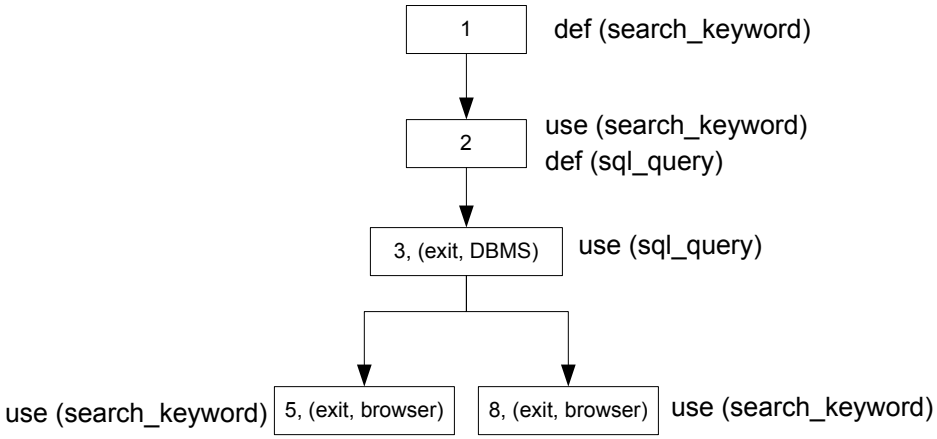


Figure 3.2 A CDG for search_keyword

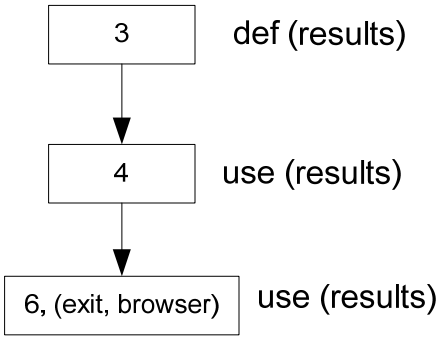


Figure 3.3 A CDG for results

These four exit points show that four possible EIVs exist in the system; however, the amount of testing needed to determine whether the EIVs exist is not known. In the next section, a coverage criterion will be defined. This criterion will help security practitioners determine how much testing on these graphs is considered sufficient.

3.3.4 Test Data Coverage, Selection, and Execution

3.3.4.1 Coverage Criterion

A number of detailed path coverage criteria for data flow testing have been proposed (Howden 1975, Laski and Korel 1983, Ntafos 1984, Woodward et al. 1980). All proposed criteria aid the tester in selecting the most effective paths in a DEF/USE graph; however, because EIV analysis only concentrates on revealing one class of fault, a more specialized criterion is required. A path is a set of edges of the CDG that demonstrates how node B can be reached from node A. Formally, the coverage criterion for EIV analysis is defined as:

- Let P be a set of paths to be tested for a CDG.
- Let E and N_e be a set of edges and exit nodes for the CDG respectively.
- P satisfies the coverage criterion for EIV analysis if
  - $\forall n \in N_e$, $\exists p \in P$ such that n is the last node in p.
  - $\forall e \in E$, $\exists p \in P$ such that e is included in p.

Although HTTP[23] is a stateless protocol, web applications are not usually stateless; they can be stateful by using session management mechanisms such as cookies (Kristol and Montulli 2000). Attempting to access a web page while not in the right state commonly results in an error. Therefore, to reach the first node of a path to begin testing, the security practitioner needs to examine the sitemap and determine the path to reach the web page that allows the first node to be accessed.

3.3.4.2 Test Data Selection

Test data have to be carefully selected to cater to each specific exit point because each external system interprets the information differently. Input data, when passed to external systems, are categorized into two types by these external systems:

- Reserved words/characters – These are words and characters that have special meanings; they are interpreted and executed by the external systems.
- Data – The data can be classified into various data types such as String, Integer, etc.

Only reserved words/characters can modify the syntactic information passed from a web application to an external system. Hence, the security practitioner needs to select data that can satisfy one requirement:

- The data has to cause the external system to interpret the data as reserved words/characters rather than data.

Therefore the security practitioner has to examine the external system's documentations to determine how to force data to become reserved words/characters. For example, let's assume that the DBMS in Figure 3.2 is MySQL (Vaswani 2004). Upon reviewing the MySQL's documentation, the practitioner may decide that if the data is not enclosed in single quotes (such as 'data') and it matches one of the reserved words/characters then MySQL will treat the data as reserved words/characters. Hence, the practitioner can simply use three test cases to test for the path leading to the MySQL exit point:

1. The data is not enclosed in single quotes, it can simply be any reserved word/character such as *SELECT*.
2. Escape the enclosure before injecting a reserved word or character, is *' SELECT*. The single quote before *SELECT* forces the data, enclosed in single quotes, to become *'' SELECT '*which means that *SELECT* is now treated as a reserved word/character.
3. To specify special characters in the data, the MySQL manual (Widenius and Axmark 2002) states that MySQL recognizes several escape sequences; these sequences start with the backslash character \. Table 4.D.1 displays these escape characters. The table shows that a single quote character can be escaped using \'; if the web application inserts the escape character before the single quote character then the character loses

[23] http://www.ietf.org/rfc/rfc2068, last accessed February 8, 2010

its special meaning. Hence, the third test case needs to escape the escape character (Table 3.1).

Table 3.1 Escape sequences for MySQL

Escape Sequence	Character Interpreted
\0	An ASCII 0 (NUL) character
\’	A single quote (‘) character.
\”	A double quote (“) character.
\b	A backspace character.
\n	A newline (linefeed) character.
\r	A carriage return character.
\t	A tab character.
\Z	ASCII 26 (Control-Z).
\\	A backslash (\) character.
\%	A percent (%) character.
_	An underscore (_) character.

The above data is only applicable for paths that do not contain any nodes within a branch, or if there are nodes within a branch, then the branching condition is not dependent upon the input under test. If the path contains nodes that are within a branch and the branching condition is dependent upon the test input, the practitioner needs to modify the test data for this input to satisfy the coverage criteria.

3.4 Case Study

A case study on a web application was performed in order to determine the fault detection capability and efficiency of the proposed approach. The application used for this case study is a commercial application, which was initially released on April 4th, 2004. The application is a powerful search engine that allows users to search for the latest product specification data from thousands of international standards. The web application has many users around the world; the users come from a wide variety of organizations from defense departments to automobile manufacturers. The application is a typical 3-tier web application, specifically using Internet Explorer, Apache+PHP and MySQL (Williams and Lane 2002) on each tier. The web application contains ~25 KLOC. It has received eight revisions; these revisions added new features and corrected many bugs and vulnerabilities. The first six revisions were corrective maintenance and were released in the application’s first 18 months of service. Revision six involved a detailed security review (Howard and LeBlanc 2003, Lipner 2000); the security review involved the following steps:

1. All web pages of the web application were visited and parsed for inputs.
2. These inputs were then used in a penetration test.
3. The source code containing vulnerable inputs were reviewed and guards were either added or modified.
4. Steps 2 and 3 were repeated until the inputs were considered to be safe from EIV attacks.

The organization revealed that the security review took 24 person-hours to complete. The bug-tracking database used by the development team (Doar 2005) reveals 68 EIVs were found and corrected for revision six. The remaining two revisions were adaptive and perfective maintenance with minor corrective maintenance (with no new EIVs were discovered) which suggests that the application is now stable; this status was confirmed by the developers of the application. For the case study, the source code for revision five was retrieved and investigated using the testing approach proposed. In order to provide a clear reference, all EIVs reported in the bug-tracking database were verified against revision five. Any EIVs that could not be replicated for revision five were discarded because they were introduced in later revisions with the addition of new features. Because no new EIVs have been discovered after the sixth revision, the total of confirmed EIVs for revision five is 68. One security practitioner was selected to perform the case study using the steps described above.

3.4.1 Drawing the Application's Sitemap

To create the sitemap, the practitioner first examined the source files. Then a crawler (REL Link Checker)[24] was used to identify the majority of the web pages. The crawler used was not able to identify web pages linked using JavaScript code; hence, the practitioner manually generated sections of the sitemap that were inaccessible to the crawler. The completed sitemap for the application took 1 hour to create, and is shown in Figure 3.4.

[24] http://www.relsoftware.com/rlc/, last accessed February 9, 2010

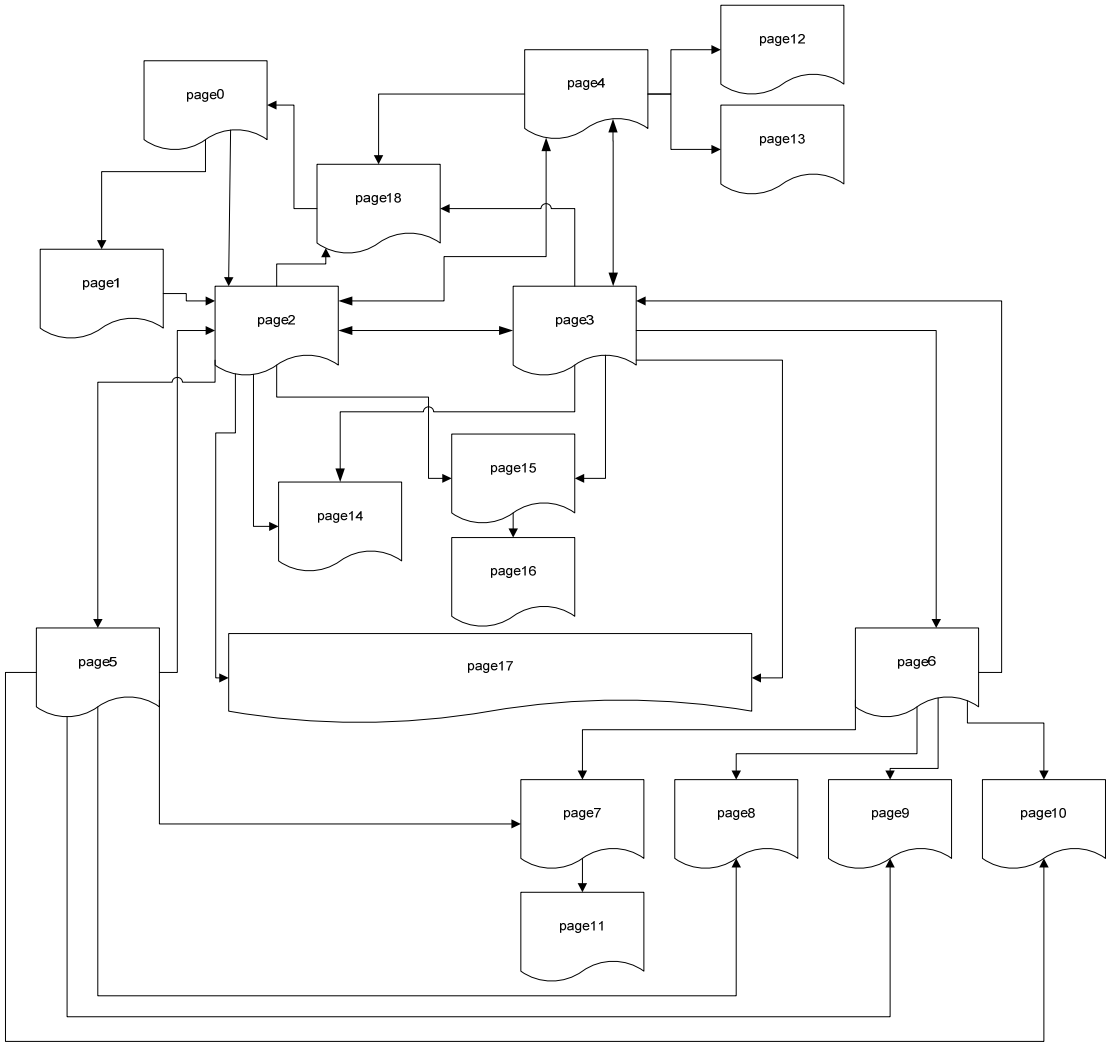


Figure 3.4 The sitemap of the application

3.4.2 Identifying the Application's Inputs

Inputs for the web application originate from two sources: the client (browser) and the MySQL database. The application is configured with *register_globals = off* (Shiflett 2004); hence, inputs originating from the client can be detected through the usage of super global arrays[25] within programming statements. Data from the MySQL database was retrieved using two function calls: *mysql_fetch_array* or *mysql_insert_id*.

Using the algorithm provided in Section 3.3.2, the Web Application Input Collection (WAIC) tool was created to aid security practitioners with this step. The tool automatically parses the source files of the web application and outputs all inter-organization and intra-organization inputs. WAIC's output contains the input type, the file, location and the input name of each identified input. The tool

[25] http://www.php.net/manual/en/, last accessed February 10, 2010

required 30 minutes to parse all the source files for inputs using an Athlon X2 3800 CPU with 2GB of RAM.

Table 3.2 Number of inputs and their sources

Input source	Number of inputs
Client/Browser	96
MySQL	545
Total	641

WAIC identified 641 inputs for the application. Table 3.2 displays the total number of inputs found and their sources. Personnel from the organization enter all of the database items. Hence, these items were initially considered as intra-organization type. To further verify this, each input was carefully examined using the available design and SRS documents. Table 3.3 shows the results from the examination.

Table 3.3 Input types

Input source	Input type	Number of inputs
Client/Browser	Inter-organization	96
	Intra-organization	0
MySQL	Inter-organization	1
	Intra-organization	544

The examination identified one inter-organization input within the database, a field that allows the customer to customize one of the display features.

3.4.3 Creating the CDGs and Choosing Test Data

3.4.3.1 Creating the CDGs

To create the CDGs, the Web Application Graph Generation (WAGG) tool was created based on the algorithm provided above. WAGG accepts the output of WAIC as its inputs. WAGG allows security practitioners to automatically generate all CDGs associated with each input identified by WAIC for the web application under test. The majority of the CDGs (99%) are very simple and contain just one exit point per graph; the graphs also do not span across more than three source files. Each line represents a node and contains a node id, previous node id, source file, line number, any DEF/USE information, and the external system if it's an exit node.

Graphs for intra-organization inputs were extremely simple, involving only one source file; hence, only three hours were required to generate all of the graphs for the intra-organization inputs. Four hours were used to create the graphs for inter-organization inputs because they were slightly more "complex". Figure 3.5 displays the most "complex" CDG that WAGG produced. Each node is labeled with the source filename, followed by the line number in brackets. Once again, filenames are obscured to ensure confidentiality. Nodes containing exit points are

labeled with the source filename, followed by the line number and the name of the external system.

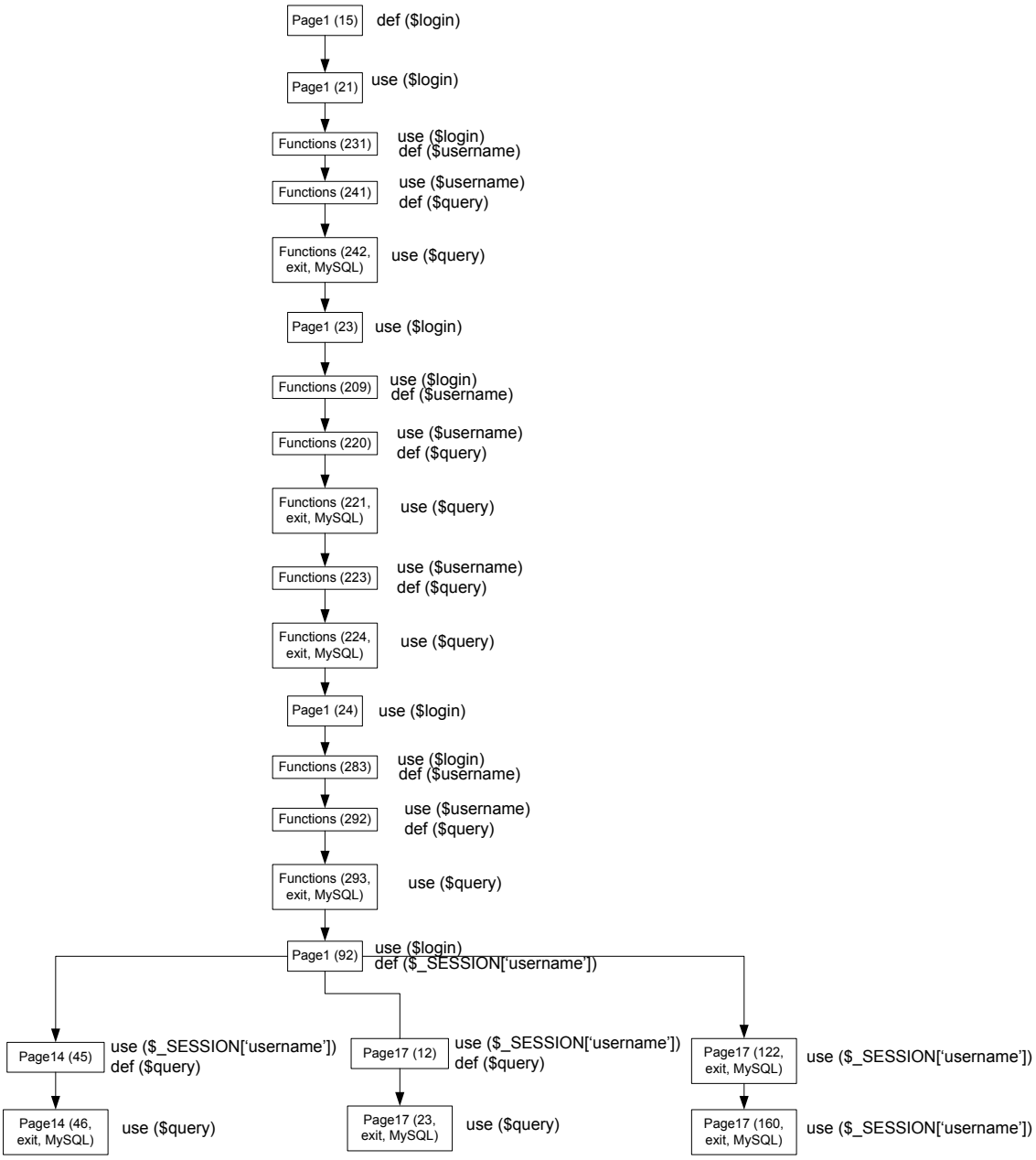


Figure 3.5 A sample CDG for the application under test

3.4.3.2 Selecting Test Data for the CDGs

For paths leading to the MySQL exit point, the practitioner used the three test cases discussed in Section 3.3.4. In terms of web browsers, this application only supports Internet Explorer. Upon reviewing all available documentation for IE 6.5, which is the lowest version supported by the application under investigation, the practitioner selected the following test data to be used for IE exit points:

- <script>alert(‘hello’)</script> - This input value attempts to insert a payload directly without any obfuscation. If the browser pops up a message box when a path is executed with this value, then the web application has an EIV.
- A - This input value will hide the JavaScript code in a harmless formatting tag. If the browser pops up a message box, after a path is executed with this value, when the mouse is moved over to the letter A, then the web application has an EIV.
- "> <script>alert(‘hello’)</script> - This input value attempts to escape the enclosure of a parameter within a tag, then closes the tag and inserts a malicious payload. For example, a benign tag like * Hello!* when expanded with the input value become <script>alert(‘hello’)</script>Hello! which means that the JavaScript code is successfully embedded. If the browser pops up a message when a path is executed with this value, then the web application has an EIV.
- " style="background:url(javascript:alert('hello'))"> - This input value also attempts to escape the enclosure of a parameter within a tag; it also obfuscates the payload code by embedding it within a style parameter. This input value will only work with IE because IE accepts JavaScript code from many uncommon tags and parameters. If the browser pops up a message when a path is executed with this value, then the web application has an EIV.

3.4.4 Test Execution, Results, and Analysis

To prioritize the test cases, the practitioner grouped the test cases according to the input type. Table 3.4 displays the test cases required for each input type.

Table 3.4 Number of paths and test cases

Input type	External system at exit point	Number of paths	Number of test cases
Inter-organization	Browser	1	4
	MySQL	103	309
Intra-organization	Browser	544	2176
	MySQL	0	0

This table shows that 648 paths should be tested; a maximum of 2489 test cases (3 test cases per path with MySQL as the exit point, and 4 test cases per path with the browser as the exit point) were executed in order for all the paths to be covered. To speed up the testing process, if one test case for a path fails (demonstrating that a EIV exists), then the practitioner simply ignored the rest of the test cases for the path.

Because the web application under investigation implements client side protection for inputs originating from the browser, a technique similar to bypass testing (Offutt et al. 2004) was used to test inputs from the browser. To test for inputs

from the DBMS, the practitioner used the MySQL command line client to insert the test values into the tables used to store data.

A capture and playback tool (AutoIt)[26] was used to aid the execution of the test cases. Only one test case per path was executed (to record the necessary key strokes and mouse clicks). The recorded scripts were then modified to change the test data to accommodate the remaining test cases. The execution process took 3 hours to complete for inter-organization test cases and 32 hours for intra-organization test cases. Table 3.5 displays the results of the tests.

Table 3.5 Test results showing the number of failed/passed paths and test cases

Data type	External System	Paths tested		Test cases	
		Passed	Failed	Passed	Failed
Inter-organization	Browser	0	1	0	1
	MySQL	29	74	87	102
Intra-organization	Browser	453	90	1816	90
	MySQL	0	0	0	0

This table reveals that the web application does not perform any input verification and validation for intra-organization inputs. As more and more web applications increase their reliance on intra-organization inputs from external systems, the number of EIVs will only increase unless developers begin to validate inputs from these external data sources.

The test results show that 165 EIVs exist (Table 3.5 shows that 165 paths failed) for revision five of the test application. However, the security review only identified 68 EIVs. The 165 potential EIVs were tested in revision six; (Table 3.6).

Table 3.6 EIVs found

Data type	Revision 5		Revision 6		Revision 7		Revision 8	
	EIV analysis	Review	EIV analysis	Review	EIV analysis	Review	EIV analysis	Review
Inter-organization	75	68	7	0	7	0	7	0
Intra-organization	90	0	90	0	90	0	90	0

Upon discussions with the developers, the application was found to be highly susceptible to intra-organization inputs because it assumes all intra-organization inputs are inherently safe. While intra-organization inputs were not examined during the security review process and hence they were not detected by the security review, the remaining 7 inter-organization EIVs should have been identified and addressed. When presented with the results, the organization

[26] http://www.autoitscript.com/autoit3/, last accessed February 9, 2010

revealed that the approach they used was not able to identify the 6 inter-organization inputs originating from JavaScript rather than the common form fields. The 7th EIV detected resulted in a stored XSS vulnerability. This vulnerable inter-organization input was code reviewed; however, because the input is transmitted to a MySQL server to be stored rather than being printed to the browser, the code review process examined the guard for the MySQL exit point rather than the guard for the browser exit point. Therefore, the EIV was not detected during the security review. The developers have confirmed that the additional EIVs discovered using this approach are valid; they have addressed all the EIVs found in a recently released revision of the web application. When the test cases were re-applied to this new revision, no EIVs were detected.

Table 3.7 Effort

Technique	Security Review	EIV Analysis	
Input Space	Inter-organization inputs only	Inter-organization inputs	Intra-organization inputs
Time required	**24 hours**	**7.5 hours**	37.5 hours

Table 3.7 shows the effort required for the security review and EIV analysis. The total effort required for EIV analysis is not a sum of the intra and inter organization effort because the effort for the sitemap creation and input identification are shared. Although the security review took only 24 hours to complete, it did not consider intra-organization inputs. If EIV analysis did not examine intra-organization inputs, then the testing process would only require 7.5 hours to complete; and it identified 7 additional EIVs than the security review process. This means, for this case study, EIV analysis can reduce the required time to perform a security review by 69%. Finally, the security review process was penetration testing with a patching component. Penetration testing uses a "librarian testing" approach which simply attempts to exploit known EIVs on a new application (Thompson 2003). Unlike penetration testing, EIV analysis is a testing strategy designed to discover EIVs; it is a technique belonging in the "unanticipated user input" class of techniques (Whittaker and Thompson 2003).

3.5 Related Work

Many techniques and approaches to detect, or mitigate against, vulnerabilities have been proposed. In this section, these techniques are briefly presented and discussed.

Many techniques address an individual class of web application vulnerability. These techniques often concentrate on one popular vulnerability type: SQL injection. SQLrand (Boyd 2004), AMNESIA (Halfond and Orso 2005), SQL-Guard (Buehrer et al. 2005), SQLCheck (Su and Wassermann 2006), CSSE (Pietraszek and Berghe 2005), WASP (Halfond et al. 2006) are all approaches aimed at addressing SQL injection vulnerabilities. SQLRand inserts random tokens into SQL statements and uses a proxy server to translate these tokens. An

incorrect query can be detected if the SQL query does not contain the correct tokens. This approach, while effective, can be defeated if the randomized tokens can be guessed; it is also complex to setup with the addition of the proxy server. AMNESIA, SQLGuard and SQLCheck are all model-based approaches. AMNESIA uses static analysis and runtime monitoring to detect for SQL injection vulnerabilities. Static analysis is used to build models of the SQL statements, while the runtime engine detects whether the query strings matches the models. This approach is prone to false negatives and positives if the static analysis used to build the model is not effective. SQLGuard requires the developers to call special functions to build a model of the SQL query to be used. SQLCheck uses a formal definition of an SQL injection vulnerability and identifies SQL injection attacks based on the formal definition. Both approaches require developers to learn and gain experience with complex models (in case of SQLCheck) or APIs (if SQLGuard is utilized). CSSE and WASP are dynamic approaches designed to address SQL injection vulnerabilities using taint analysis. These approaches attempt to mark negative tainting (CSSE) or positive tainting (WASP) to identify malicious query statements before they are passed onto the DBMS. Both approaches involve modification to either the runtime engine or usage of a specialized API; hence, deployment can be expensive or programmers need to learn yet another API respectively. While some of the approaches listed claim to support other types of EIVs (SQLCheck, CSSE), their supplied tool only concentrates on detecting one type of EIV (SQL injection) which leaves the system vulnerable to other types of EIVs. EIV analysis does not have this limitation because the strategy is designed to address all types of EIVs.

General approaches to applications' security have also been proposed which address all types of EIVs. Security Gateway proposed by Scott and Sharp (2002) is an application firewall that filters out all malicious inputs before they reach the web application. The effectiveness of this approach is dependent on an administrator's ability to produce complex and effective rule sets. Nguyen-Tuong et al. (2005) proposed a dynamic approach to detect EIVs through taint analysis. This approach requires the runtime engine to be modified which causes complex deployment and increased overhead.

All approaches discussed are application security techniques. That is, they protect the software after it has been built (McGraw 2004). EIV analysis is a software security strategy; the approach increases the security of web applications during the development process and before they are deployed on live servers. Several software security approaches related to EIV analysis currently exist. They can be classified into two categories: static analysis approaches and black-box testing (Beizer 1995) techniques.

Static approaches have been used to detect vulnerabilities with some success. Shankar et al. (2001) proposed a static approach that can detect format-string vulnerabilities commonly found in C-based applications. The method defines two extended data types, tainted and untainted, which help reduce the number of false

positives generally associated with static analysis methods. Zhang et al. (2002) and Johnson and Wagner (2004) further extend the approach by using it to assess security issues with the Linux Security Modules framework and user/kernel pointers successfully. These approaches are designed to detect vulnerabilities in C-based applications, and hence their effectiveness with scripting languages such as PHP, Ruby, and Python remain unknown.

Although static analysis is a well known technique, approaches that specifically target web applications' EIVs are not common. Proposed approaches such as those by Livshits and Lam (2005), Martin et al. (2005), Balzarotti et al. (2007), WebSSARI (2004) and Pixy (2006) have limitations. Techniques proposed Livshits and Lam (2005), Martin et al. (2005) are designed specifically for SQL injection vulnerabilities, and hence it cannot be used to detect other EIVs. Balzarotti et al. (2007) presents a static analysis approach capable of detecting both workflow attacks and data-flow attacks. However, the approach cannot detect all EIVs. For example, many websites now have multiple web applications sharing the same database. An attacker can utilize a vulnerability in one web application (A) to inject a payload into the database which will then be used by the other web application (B). If the approach is used to analyze (B), this vulnerability would be undetected. WebSSARI does not model conditional branches that result in many false positives. Furthermore, the WebSSARI tool is not available and hence, no comparison with it can be made. Pixy is an advanced static taint analysis tool available for PHP. However, attempts to use the tool for comparison with EIV analysis reveal several issues:

- Pixy cannot detect stored XSS, and other types of EIVs (OS/Filesystem and Interpreter interactions).
- Six of the 7 XSS vulnerabilities it detected, when used on the case study's application, are false positives.
- It ignores path information and tainted data inside objects, and hence, its reports contain false positives and negatives.
- It requires a very large amount of memory to model SQL injections. In fact, on the test machine which has 2GB of RAM, it crashed repeatedly when used on the case study's application.

Offutt et al. (2004) proposed a black-box testing approach that requires a customized client to test web applications. The customized client allows the tester to bypass all client side protection mechanisms; and hence, if a web application is dependent on client side verification of inputs, it will fail the test cases. QED (2008) and Ardilla (2008) attempt to generate SQL Injection and XSS attacks automatically. However, QED cannot target second order XSS attacks and requires users to learn a custom specification language. Ardilla suffers from low code coverage and a 42% false positive rate. Secubat (2006) and other commercial web scanners such as Acunetix Web Vulnerability Scanner[27] extend bypass testing by creating tools that provide automatic penetration testing

[27] http://www.acunetix.com, last accessed February 7, 2010

for web applications without using the web applications' target clients. Commercial applications are proprietary and closed source; hence they cannot be examined in detail. Secubat currently has no plug in to detect all types of XSS; for example, stored XSS. Lin and Chen (2006) extend traditional black-box testing techniques with elements of static analysis by including a tool to automatically inject guards at input points found through the crawling component. This approach does not guarantee correctness of the modified program and hence, the modified program may not meet the original requirements. All black-box testing approaches for web applications have a limitation that not all inputs can be detected through web page parsing (Offutt et al. 2004); hence, only an approximation of the inputs is possible.

While many black-box approaches to web application security testing have been proposed, no white-box strategies have been presented. EIV analysis is a white-box approach that utilizes data flow graphs to test for EIVs. Just as other software security approaches, EIV analysis allows a company to test its web applications before they are launched. EIV analysis can also coexist with all of the approaches presented. That is, an organization can use static analysis approaches to automatically identify some EIVs, then use EIV analysis and black-box approaches to locate additional vulnerabilities. Finally, application security approaches can be applied to monitor the web application when it is deployed.

This page is intentionally left blank.

Chapter 4 – Automatic Identification of Web Attacks

Network intrusion detection systems (NIDS) are often classified as either misuse or anomaly based systems (A-NIDS). Misuse based systems contain rules designed to filter out known attacks. One popular misuse based system is SNORT[28] which is used by over 270,000 users. Misuse systems cannot detect and prevent attacks that are not contained in the rule set because these attacks are too recent (zero day attacks); hence, most new NIDS approaches are anomaly based. A-NIDS do not rely on any rule set; therefore they can potentially detect these new attacks (Forrest et al. 1996, Anderson 1972, Heberlein et al. 1990). With A-NIDS, empirical information on system usage is first collected. Using collected information, the A-NIDS creates a model of normal behaviour. Observations that deviate from the model are classified as anomalous.

A-NIDS often utilize machine learning (ML[29]) techniques. Lazarevic et al. (2005) and Tsai et al. (2009) provide a review of existing ML based A-NIDS. Traditional A-NIDS concentrate on low-level packet information implying that application specific information is lost (Krugel 2002). As a result, A-NIDS often have low detection rates for attacks targeting the web application layer. A new generation of A-NIDS has been proposed to specifically target the web application layer; a brief overview of these A-NIDS follows.

Kruegel et al. (2003, 2005) presented one of the first A-NIDS designed specifically for web applications. The system contains six anomaly models and six techniques for estimating the probability of an attack based upon these models. Valeur et al. (2005) presents an approach that profiles normal database access performed by web applications to detect SQL injection attacks on a DBMS. Swaddler (Cova et al. 2007a) extends Kruegel et al. (2003, 2005) by also examining the state of the web application.

Ingham et al. (2006) introduces Deterministic Finite Automata induction as a method to detect malicious web requests. However, their results show that the approach currently suffers from low detection and high false positive rates. Cheng et al. (2008) proposes an Embedded Markov Model to detect attacks and monitor users' behaviour. Estevez-Tapiador et al. (2005) uses a hybrid approach that is both learning and specification-based. The approach builds a Markov model using the specification of the HTTP protocol and the actual payload from the training data. Sphinx (Bolzoni and Etalle 2008) detects attacks on web application data flows using "positive signatures" which are rules that match normal inputs rather than malicious inputs. Park and Park (2008) uses an extended Needleman-Wunsch (1970) algorithm to build a profile of normal web requests. Future web requests that do not match this profile are classified as anomalous.

[28] http://www.snort.org/, last accessed January 8, 2010.
[29] ML in this dissertation is an acronym for Machine Learning.

All A-NIDS, often using ML algorithms, classify data as either malicious or benign (Tsai et al. 2009). Current A-NIDS for web applications attempt to create custom anomaly models for this classification step. However, no existing A-NIDS for web applications leverage the available knowledge from various ML techniques; and traditional ML-enabled NIDS suffer from low detection rates because they have no domain knowledge of the application layer. This chapter presents a novel A-NIDS for web applications called The Automatic Identification of Web Attacks System (AIWAS). This approach differs from the available techniques because it does not create an anomaly model. It creates a model of the HTTP input; this application-level model in conjunction with captured traffic allows AIWAS to learn "normal" HTTP traffic patterns for individual applications.

The remaining sections of this chapter are organized as follows: Section 4.1 introduces AIWAS in detail. Section 4.2 presents a case study to evaluate the effectiveness of AIWAS.

4.1 AIWAS

AIWAS is an intrusion detection system specifically for the web application layer. AIWAS is an A-NIDS that classifies future system usage into benign or malicious categories without relying on signatures. This learning-based approach has an advantage that it is trained on a per-website basis, providing effective localization. That is, each website will contain different usage profiles, which learning-based approaches can recognize.

AIWAS is a learning-based system comprising of two distinct components: the Sentinel and the Oracle. AIWAS can be operated in detection (alerting system administrators to potential attacks) or prevention mode (blocking requests identified as malicious). Figure 4.1 shows the modified architecture of a web application to include AIWAS.

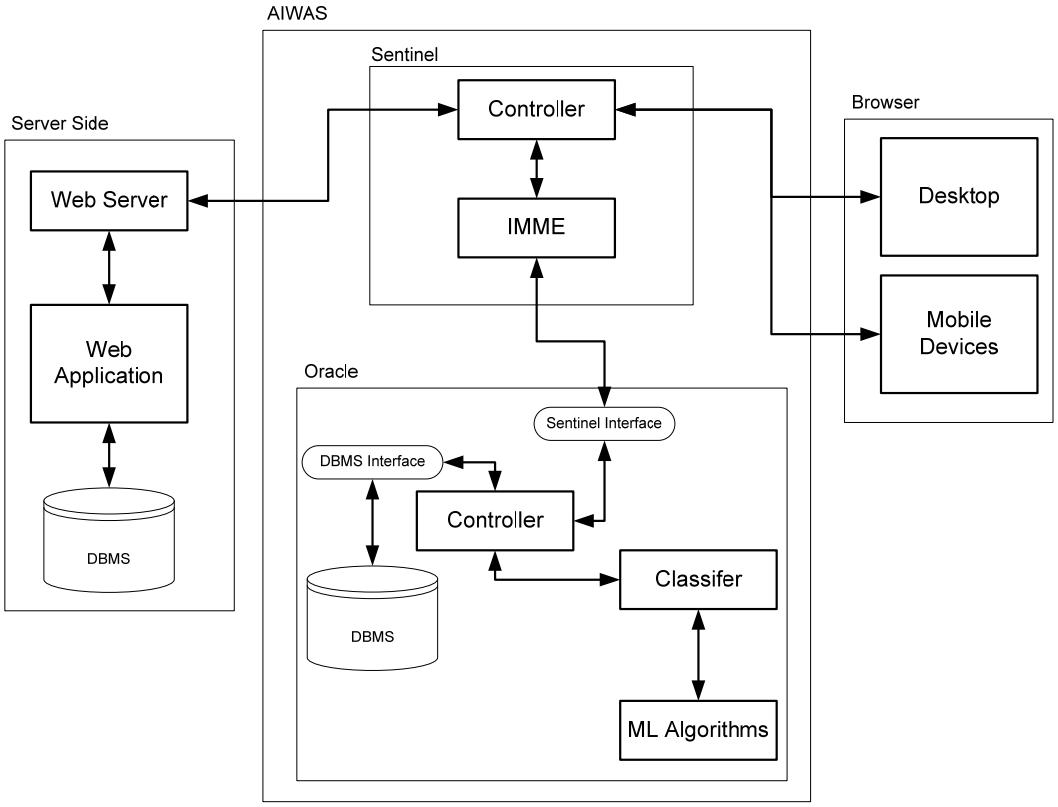


Figure 4.1 AIWAS Architecture

The Sentinel examines the dataflow between the web application and the browser. When a request is received from the browser, the Sentinel will map this request onto an instance model (IM). This IM is used as an input to the Oracle. The Oracle classifies whether the IM is malicious or benign. If the IM is malicious, the Sentinel will either reject the request or notify the system administrator.

4.1.1 Instance Model

OWASP (2010) and Cova et al. (2007b) state that the most common web application security weakness is the failure to properly validate input from the client or environment. Chapter 2 provides empirical evidence that many attacks are based upon the failure to validate inputs. Additionally, statistics on web vulnerabilities[30&31] indicate that the majority of vulnerabilities contained in web applications, such as Cross-Site Scripting (XSS) and SQL Injections, are exploited through the manipulation of input values. In fact, this is also the primary approach that commercial[32&33&34], open source[35&36], and research-based

[30] http://www.sans.org/top-cyber-security-risks/, last accessed February 3, 2010
[31] http://projects.webappsec.org/Web-Application-Security-Statistics, last accessed February 3, 2010
[32] http://www.acunetix.com/, last accessed February 7, 2010
[33] https://h10078.www1.hp.com/cda/hpms/display/main/hpms_content.jsp?zn=bto&cp=1-11-201-200^9570_4000_100__, last accessed February 2, 2010
[34] http://portswigger.net/scanner/, last accessed February 2, 2010
[35] http://www.cirt.net/nikto2, last accessed February 3, 2010

(McAllister et al. 2008, Kals et al. 2006, Antunes and Vieira 2009) vulnerability scanners use to detect web application vulnerabilities. Hence, the IM for AIWAS specifically models input values in order to allow ML algorithms to classify them effectively.

4.1.1.1 Modeled Data

A web application receives its inputs through HTTP requests. Hence, the IM should contain all the information of the HTTP request structure necessary to perform the classification. One element of the HTTP request that should be modeled is the resource requested along with its parameters; cookies are also treated as parameters. The values of these parameters are validated against several properties: data syntax, length of the data based on the specification, and character sets (OWASP 2010).

Modeling these properties is sound because these properties have been examined in the past to test systems for faults. Boundary value analysis, equivalent partitioning and fuzz testing are popular methods for selecting input values; these can be seen as value manipulation methods used to reveal software faults. Similarly, web application attackers can manipulate these properties of parameter values to reveal security vulnerabilities. In fact, this technique for attacking web applications has been discussed in the past (Offutt et al. 2004, Tappenden et al. 2006, Scambray et al. 2006, Sutton et al. 2007).

The IM should not include unnecessary information that decreases the efficiency of the ML algorithms. That is, information contained in the structure of HTTP requests, which is not indicative of an attack can make the volume and variety of information too complex for ML algorithms to analyze. For example, an HTTP request will always contain an HTTP method such as GET, POST, HEAD, PUT, etc. The IM can ignore this data because it does not change between malicious and benign inputs.

Based on the above discussion, the (generic) IM is:

$$IM = \{R, <P_{1\ length}, P_{1\ has\_non\_alpha}, P_{1\ has\_reserved\_words}> \ldots <P_{n\ length}, P_{n\ has\_non\_alpha}, P_{n\ has\_reserved\_words}>\}$$

where the attributes are:

- R: resource requested
- P: parameter (including cookies) associated with the resource requested
- n: number of parameters
- length: the length of the parameter

[36] http://wapiti.sourceforge.net/, last accessed February 3, 2010

- has non-alpha: whether the parameter value contains any non-alphanumeric characters (encoded alphanumeric characters are also treated as non-alphanumeric characters).
- has reserved words: whether the value for the parameter contains any reserved words used in the web application, such as words that are part of the programming language used to create the application, or SQL statements. For this research, because the applications used in the case study are PHP based which utilize a MySQL back-end for data storage, reserved words are defined as HTML tags and reserved words in SQL and PHP.

System administrators with considerable knowledge of their web applications can further improve the model by adding or modifying the model to be more specific. For example, instead of a generic list of reserved words for the "has reserved words" attribute, if the system administrator knows that certain words are 100 percent safe for use in the system, those words can be removed from the generic list. This allows the model to be more specific which can lead to a higher detection rate without increasing the false positive rate.

The next two sections provide an example of a mapping for normal usage and an example of a mapping for an attack.

4.1.1.2 Example of the Mapping

When the Sentinel receives a request from the browser, it transforms the IMME to the request into an IM. The following figure is an example of the raw data for a login form that the Sentinel receives.

```
[Thu Aug 14 09:55:36 2008]:  dumpio_in (data-HEAP): 37 bytes
[Thu Aug 14 09:55:36 2008]:  dumpio_in (data-HEAP): POST /folder/ login.php? HTTP/1.1\r\n
[Thu Aug 14 09:55:36 2008]:  dumpio_in (data-HEAP): 46 bytes
[Thu Aug 14 09:55:36 2008]:  dumpio_in (data-HEAP):
login=username&password=password&Submit=Login
```

Figure 4.2 An example of the request data

This is a POST request for the resource *login.php* with three additional parameters: *login, password, Submit*. The lengths of the values for the parameters are eight, eight, and five characters respectively. The values of the attributes do not have any non-alphanumeric characters or reserved words.

The IMME would transform this request into the following IM: *{login.php, <8,false,false>,<8,false,false>,<5,false,false>}*

This section showed the mapping of a normal HTTP request to an IM. The next section provides an example of a malicious IM.

4.1.1.3 Example of an Attack

Many web applications have specifications for their input fields. For example, the Canadian Imperial Bank of Commerce's online banking system specifies that passwords must be 6 to 12 alphanumeric characters[37]. However, when attackers attempt to gain access to the system, these requirements are not followed. Figure 4.2 shows an example of an HTTP request data for a login form. If an attacker attempts SQL injection attacks on this login form, the data would be different from what is shown. Figure 4.3 shows a possible SQL injection attempt to launch a stored XSS attack.

Thu Aug 14 09:55:36 2008]: dumpio_in (data-HEAP): 37 bytes
[Thu Aug 14 09:55:36 2008]: dumpio_in (data-HEAP): POST /folder/login.php? HTTP/1.1\r\n
[Thu Aug 14 09:55:36 2008]: dumpio_in (data-HEAP): 1299 bytes
[Thu Aug 14 09:55:36 2008]: dumpio_in (data-HEAP): login=username&password=a
;DECLARE%20@S%20CHAR(4000);SET%20@S=CAST(0x4445434C41524520405420766172
6368617228323535292C4043207661726368617228343030302920444543 4C415245205461626C
655F437572736F7220435552534F5220464F522073656C65637420612E6E616D652C622E6E61
6D652066726F6D207379736F626A6563747320612C737973636F6C756D6E7320622077686572
6520612E69643D622E696420616E6420612E7874…;&Submit=Login

Figure 4.3 An example of the request data

This example shows an advanced form of an SQL injection attack. The actual payload has been shortened. The attack involves using SQL reserved words and encoded characters to dynamically construct an INSERT SQL statement. The INSERT SQL statement is used to store a XSS payload. The IM for this request is: *{login.php, <8,false,false>,<**1261,true,true**>,<5,false,false>}*

The IM for this attack differs from the benign IM seen in the previous example. The value for the `password` parameter is used to inject a malicious payload, and the IM shows this. The `length` attribute for this parameter is 1261 characters, which is significantly longer than the common length for a password. Furthermore, the value contains both non alphanumeric characters, such as `;`, `%`, `@`, and reserved words for SQL like `SET` and `DECLARE`.

In essence, AIWAS attempts to learn the input specification associated with "normal" usage of the system, and validates any given input against this specification. Via the IM model, AIWAS only attempts to learn the part of the specification that can be compromised. It should be noted that AIWAS will only validate inputs to this "normal" usage specification and not the exact specification as defined in the Specification documentation. Effectively, AIWAS attempts to automate the "validation of the input from the client or environment" removing programmer errors and effort from the process. Hence, any limitation of AIWAS to completely learn the can-be-compromised component of the input specification

---

37

http://cibc.intelliresponse.com/public/en/index.jsp?requestType=NormalRequest&interfaceID=8&id=-1&source=1&question=%20what%20do%20i%20do%20if%20i%20can%27t%20sign%20on, last accessed January 12, 2010

can be viewed as being analogous to limitations by programmers to completely validate the input from the client or environment. Clearly, both AIWAS and programming attempts can be deployed together to provide a security in depth approach.

4.1.2 ML Algorithms

AIWAS requires the algorithms to classify the requests as either benign or malicious. Hence, only supervised learning algorithms can be used. The case study in this chapter will demonstrate whether usage of different ML algorithms will have significant impact on the performance of AIWAS. Four different ML algorithms are selected for this empirical investigation.

Selection of the first two algorithms, Naïve Bayes (John and Langley 1995) and Random Forests (Breiman 2001), is based on general acceptance of the efficacy of these algorithms. The last two, Rotation Forrest (Rodriguez et al. 2006) and Simple Logistic (Summer et al. 2005), can produce better results than Naïve Bayes and Random Forests (Kuncheva and Rodriguez 2007, Landwehr et al. 2005) although the number of samples which illustrate this conjecture is still limited.

4.1.3 Data Set

The DARPA 1999 data set is commonly used to evaluate IDS (Lippmann et al. 2000). However, the DARPA 1999 data set suffers from several flaws and artifacts (Mahoney and Chan 2004, McHugh 2000a, McHugh 2000b). Furthermore, the data set is not representative of current attacks on web applications. For these reasons, the DARPA 1999 data set is not used to evaluate the proposed system. Instead, three web applications are selected, with two having known vulnerabilities. Once trained using generated training data sets, AIWAS was evaluated on its ability to detect attacks created from the known vulnerabilities.

As with other learning based approaches, AIWAS requires a training data set before it can begin classifying live requests. Obtaining a representative training data set is a challenge system administrators will face when using AIWAS. This section discusses the various sources that system administrators can use to obtain the necessary training data.

4.1.3.1 Set AIWAS to Learning Mode on Live Server

One method of obtaining a training data set is to set AIWAS to learning mode. While in this mode, AIWAS will collect all requests and store the IM equivalent. This method allows system administrators to obtain actual usage data from the web application and use them as the training data set. Hence, the training data set is the most representative of the live requests that AIWAS will encounter. However, this method is time consuming because there is a waiting period while AIWAS collects the data. Furthermore, system administrators will need to be experts at classifying the IMs.

4.1.3.2 Using Existing Server Logs

System administrators can also use existing server logs as a source to create the training data set. However, most server logs do not contain POST request data by default. Hence, system administrators need to ensure that the server logs chosen do contain all the necessary data. Additionally, just like 4.1.3.1, the IMs obtained from this approach also need to be classified.

4.1.3.3 Simulate Normal and Malicious Usage with a Small Subset of Users

This approach is used in this study to generate the training data set. The approach involves the following steps:

1. Set up the web application and AIWAS in a closed environment with the same settings as the production system.
2. Set AIWAS to learning mode.
3. Allow test users to use the system normally.
4. Classify all IMs obtained from Step 3 as "benign".
5. Ask test users to use well known attack techniques to attack the system.
6. Classify all IMs obtained from Step 4 as "malicious".
7. Combine IMs from Steps 4 and 6 into one training set.

This approach requires users with knowledge vulnerability attack techniques to generate the malicious data. Furthermore, the training data set may not be a perfect representative of actual usage data. However, the approach does not require manual classification of IMs.

4.2 Case Study

In order to determine the effectiveness of AIWAS, a case study was performed on three web applications:

1. A proprietary commercial web application currently being deployed (WA1). This web application has no known/published vulnerabilities.
2. Phd Help Desk[38], an open source ticket support system. This application has 11 known vulnerabilities as posted on the OSVDB.org website.
3. OpenDocMan[39], an open source document management system. This application has 13 known vulnerabilities as posted on the OSVDB.org website.

The first application is selected to evaluate the effectiveness of AIWAS on a commercial system versus commonly used open source applications. Due to the proprietary nature of the application, no vulnerabilities are known for a test data set; hence, the evaluation is performed using the standard stratified 10-fold cross-validation approach (Witten and Frank 2005).

The selection of the two open source systems for testing was based on the following two criteria:

[38] http://www.p-hd.com.ar/, last accessed January 19, 2010
[39] http://www.opendocman.com/, last accessed January 19, 2010

1. They should not be popular and well known web applications. In other words, the personnel responsible for generating the training data set should not be familiar with the existing vulnerabilities for these applications. Additionally, vulnerability scanners should not have these existing vulnerabilities in their database. If either condition exists, the training data set will be bias towards detection of these vulnerabilities.
2. They must have known and published vulnerabilities. Once the training data set is generated and AIWAS has been trained, the web applications will be attacked with these known vulnerabilities. This approach allows AIWAS' effectiveness against real attacks to be evaluated.

The training data sets for all three applications were obtained using the approach discussed in Section 4.1.3.3. To evaluate AIWIAS, receiver operating characteristics (ROC) graphs are used. Provost and Fawcett (2001) introduced ROC graphs to machine learning as a method of visualizing classifiers' results; the ROC curve can be seen as a bi-dimensional representation of the classifiers' performance (Fawcett 2003).

In order to determine whether class imbalance, which is often associated with intrusion detection systems (Chawla et al. 2004), will affect AIWAS, the training data sets were used as is, and with the Synthetic Minority Over-sampling TEchnique (SMOTE) (Chawla et al. 2002). SMOTE has been demonstrated to provide better performance on unbalanced data sets when tested using C4.5, Ripper, and Naïve Bayes (Batista et al. 2004). Although Weiss and Provost (2003) observed that naturally occurring distributions are not always the optimal distribution, there are no standards as to what the ratio should be for web systems. Hence, the commonly accepted ratio of 1 to 1 ratio between malicious and benign IMs is used for the SMOTE balanced training data sets.

4.2.1 Results – 10-fold cross validation

Figures 4.4 to 4.15 show the ROC curves for all four algorithms with the training data set without SMOTE. These ROC curves allow visual analysis of costs (penalties) associated with false-negative errors versus false-positive errors. Analysis of the curves shows that the "corners" of the curves are close to the upper left corner indicating the effectiveness of the algorithms.

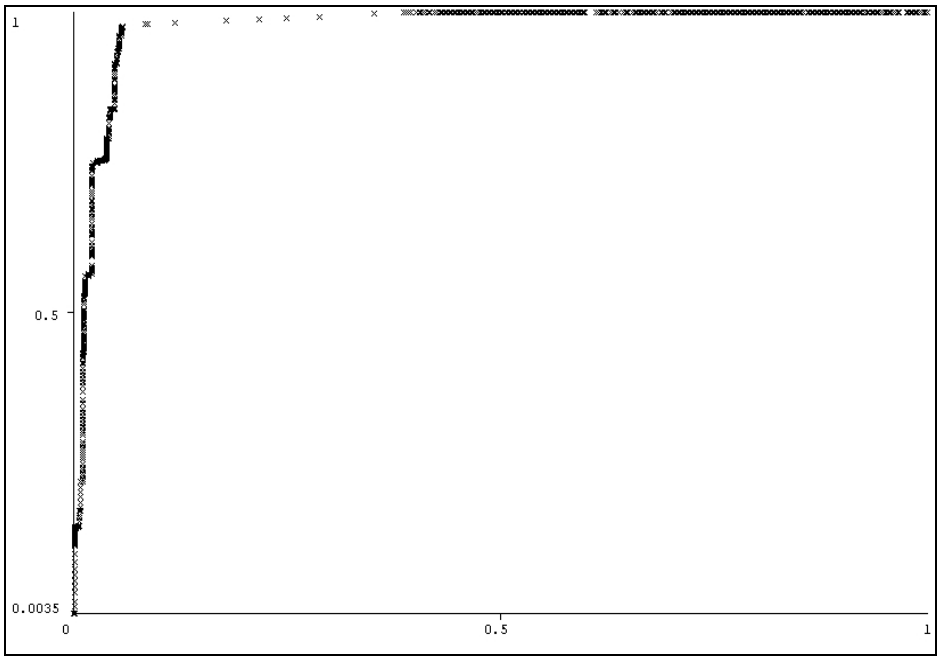


Figure 4.4 10-Fold Cross Validation ROC Curve for WA1 with Naïve Bayes

Figure 4.5 10-Fold Cross Validation ROC Curve for WA1 with Random Forest

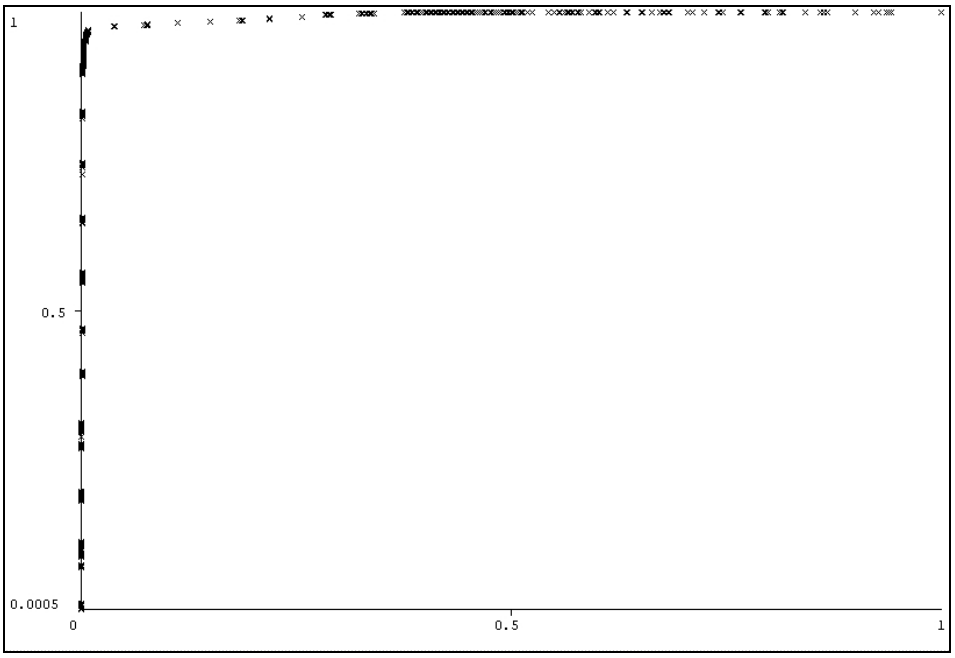


Figure 4.6 10-Fold Cross Validation ROC Curve for WA1 with Rotation Forest

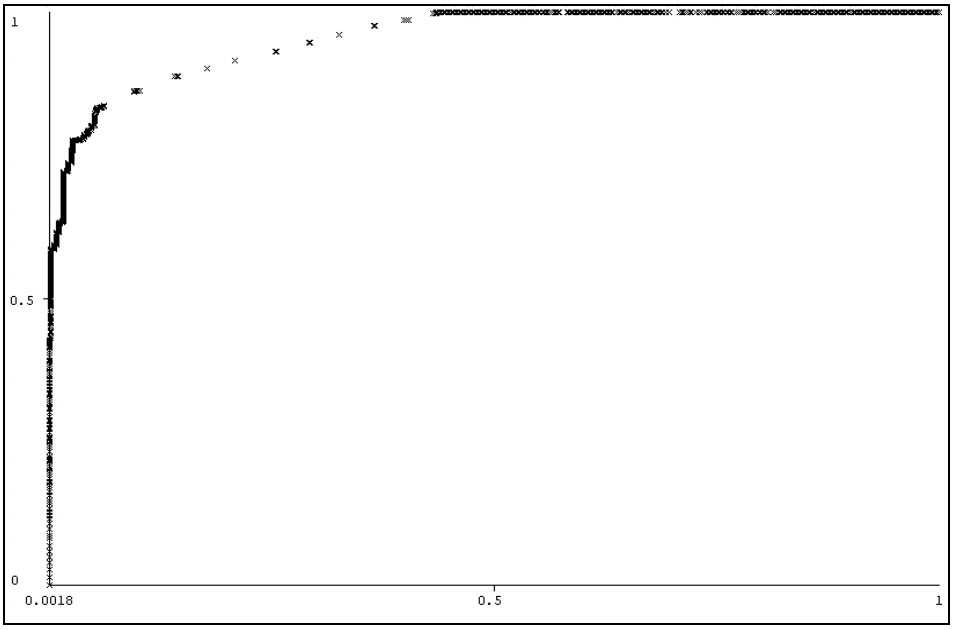


Figure 4.7 10-Fold Cross Validation ROC Curve for WA1 with Simple Logistic

Figure 4.8 10-Fold Cross Validation ROC Curve for Phd Help Desk with Naïve Bayes

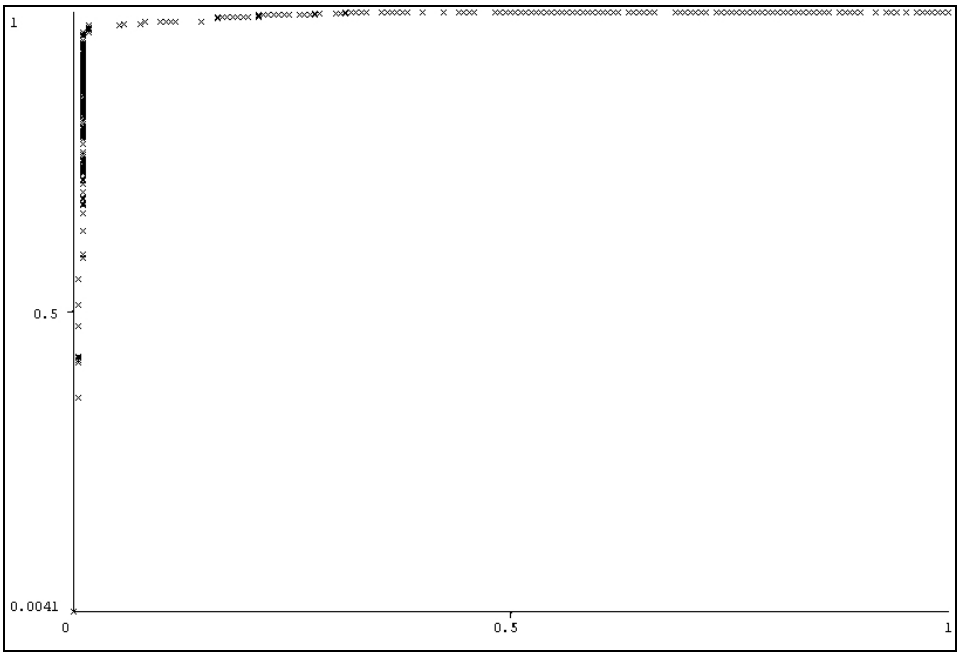


Figure 4.9 10-Fold Cross Validation ROC Curve for Phd Help Desk with Random Forest

Figure 4.10 10-Fold Cross Validation ROC Curve for Phd Help Desk with Rotation Forest

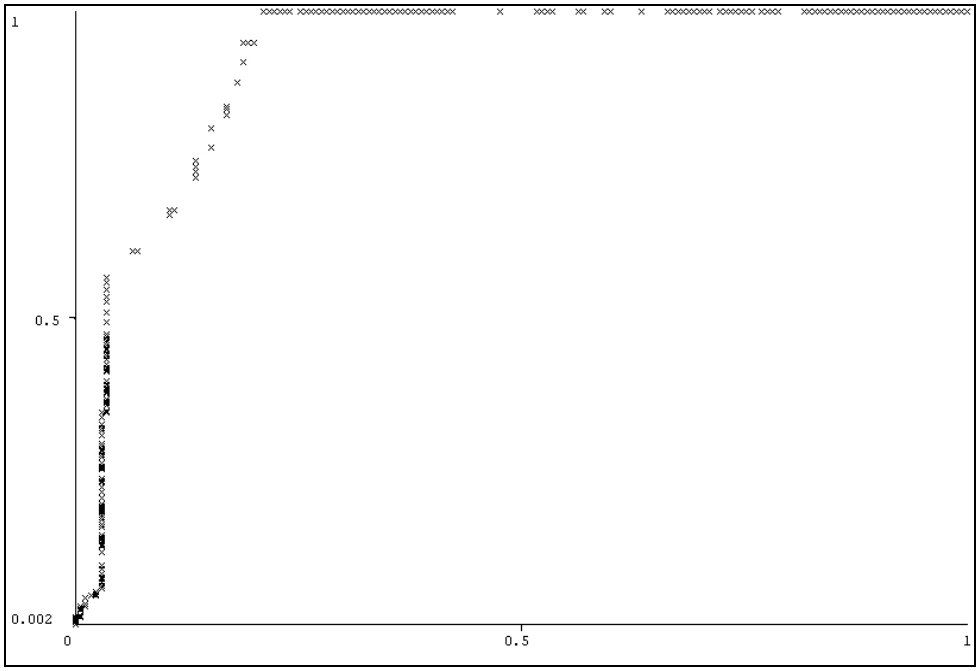


Figure 4.11 10-Fold Cross Validation ROC Curve for Phd Help Desk with Simple Logistic

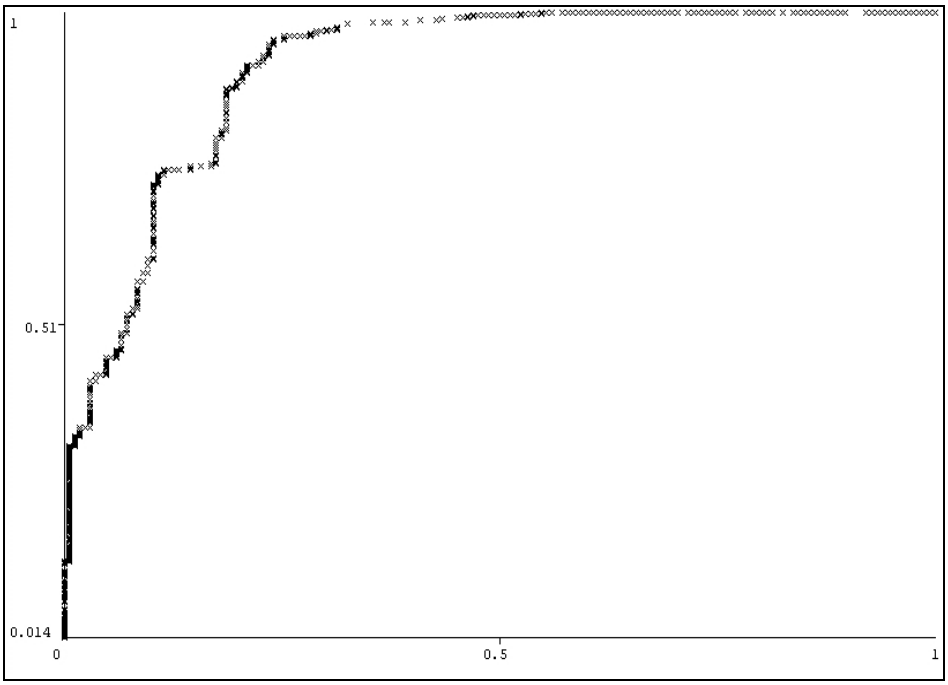


Figure 4.12 10-Fold Cross Validation ROC Curve for OpenDocMan with Naïve Bayes

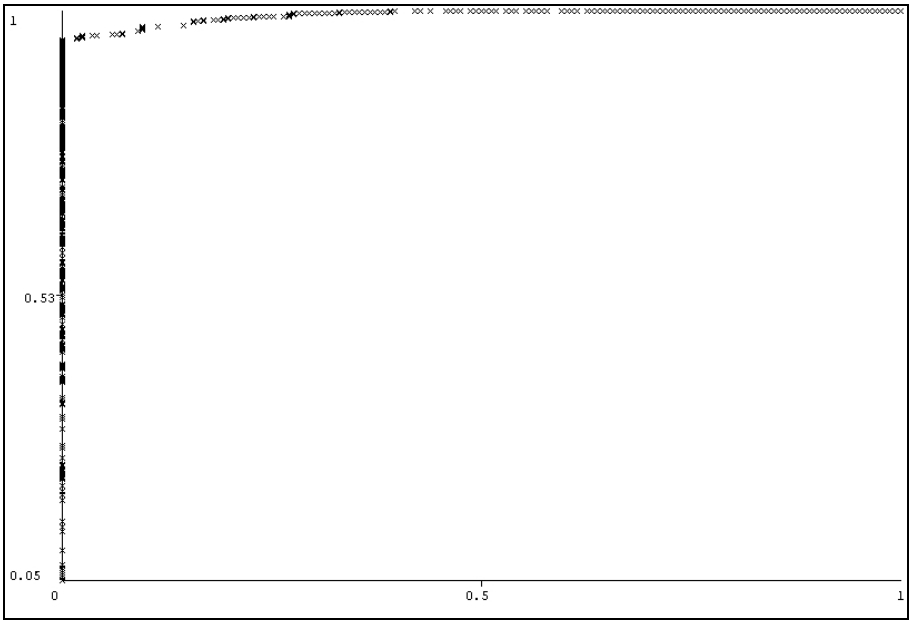


Figure 4.13 10-Fold Cross Validation ROC Curve for OpenDocMan with Random Forest

Figure 4.14 10-Fold Cross Validation ROC Curve for OpenDocMan with Rotation Forest

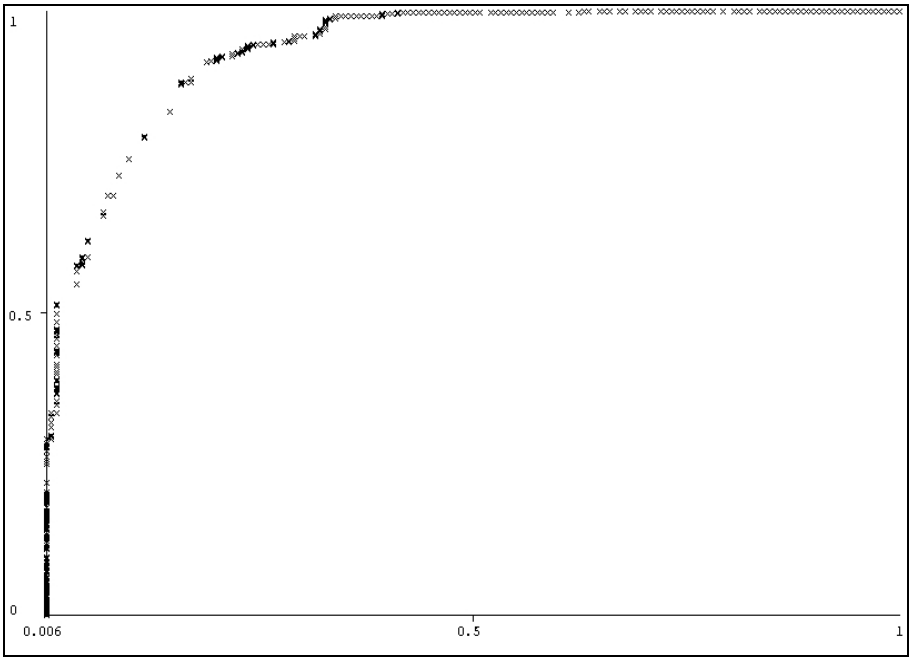


Figure 4.15 10-Fold Cross Validation ROC Curve for OpenDocMan with Simple Logistic

Table 4.1 presents the comparison of the "corner" of the ROC curves. No unique method exists for comparing ROC curves, hence five metrics for this comparison are used:

- Precision which measures the exactness of the system;
- Recall which measures the completeness of the system;
- F-measure which is derived from the Precision and Recall;

- Kappa - Cohen's Kappa statistic (Rourke et al. 2001) which measures the chance-corrected agreement between the actual and predicted classification), and
- Matthew's Correlation Coefficient (MCC) (Baldi 2000) which is equivalent to Pearson's correlation coefficient for dichotomous data.

The figures in this table show that AIWAS' accuracy rate varies depending on which algorithm is used. However, the results also show that the IM, presented in Section 4.1, allows all four algorithms to classify the HTTP requests effectively.

Table 4.1 Accuracy metrics

		WA1	Phd Help Desk	OpenDocMan
Naïve Bayes	Precision	0.923	0.938	0.889
	Recall	0.915	0.932	0.873
	F-measure	0.907	0.928	0.854
	Kappa	0.710	0.805	0.543
	MCC	0.741	0.821	0.611
Random Forest	Precision	0.920	0.923	0.891
	Recall	0.911	0.914	0.873
	F-measure	0.903	0.907	0.853
	Kappa	0.694	0.746	0.546
	MCC	0.728	0.772	0.609
Rotation Forest	Precision	0.948	0.970	0.971
	Recall	0.948	0.970	0.972
	F-measure	0.946	0.970	0.972
	Kappa	0.838	0.921	0.918
	MCC	0.842	0.920	0.918
Simple Logistic	Precision	0.913	0.939	0.917
	Recall	0.904	0.933	0.914
	F-measure	0.894	0.930	0.907
	Kappa	0.666	0.810	0.720
	MCC	0.704	0.825	0.740

The four ML algorithms were tested again, this time using the SMOTE balanced training data sets. The results from the comparison of the "corners" of the ROC curves can be seen in Table 4.2. The results show that the SMOTE balanced training data sets do offer improvements over the original unbalanced training data sets. However, whether this also holds when AIWAS is used to detect actual attacks will be examined in the next section.

Table 8.2 Accuracy metrics with SMOTE

		WA1	Phd Help Desk	OpenDocMan
Naïve Bayes	Precision	0.976	0.942	0.902
	Recall	0.976	0.940	0.896
	F-measure	0.976	0.940	0.896
	Kappa	0.952	0.880	0.793
	MCC	0.952	0.882	0.798
Random Forest	Precision	0.952	0.956	0.966
	Recall	0.948	0.955	0.965
	F-measure	0.947	0.955	0.965
	Kappa	0.895	0.911	0.929
	MCC	0.899	0.911	0.931
Rotation Forest	Precision	0.969	0.968	0.979
	Recall	0.969	0.968	0.979
	F-measure	0.969	0.968	0.979
	Kappa	0.938	0.936	0.959
	MCC	0.938	0.936	0.958
Simple Logistic	Precision	0.945	0.944	0.905
	Recall	0.939	0.942	0.905
	F-measure	0.939	0.942	0.905
	Kappa	0.878	0.884	0.810
	MCC	0.883	0.886	0.810

4.2.2 Results – Real Vulnerabilities

For this evaluation, test sets based on attacks from published vulnerabilities were used for evaluation. Based on the 11 and 13 published vulnerabilities for Phd Help Desk and OpenDocMan, 22 and 25 attacks were generated respectively. To generate extra attacks from a limited number of vulnerabilities, an approach similar to mutation testing (Offutt 1994) was used.

With dedicated test sets, two additional methods of classifying the IMs were used. Both of these methods are based on the Principal of Aggregation (Rushton et al. 1983) which states that the result from a set of multiple measurements is more stable and representative than any single measurement. The first method, Aggregate Malicious, classifies an IM as malicious when two or more ML algorithms classify it as malicious. The second method, Aggregate Benign, only classifies an IM as benign when two or more ML algorithms classify it as benign. Both of these methods will be considered as "algorithms" for the remainder of this chapter.

The ROC curves for the six algorithms are shown in Figures 4.15-4.27. Similar to Section 4.2.1, the "corners" for these curves are very close to the upper left corner implying that all approaches have a high True Positive Rate (TPR) and a low False Positive Rate (FPR). The "corners" from Figures 4.23 and 4.24 show that

the Random Forest and Rotation Forest are not as effective at detecting vulnerabilities for OpenDocMan as the other two algorithms; however, they are still close to the upper left corner. The results from Table 4.3 confirm this observation.

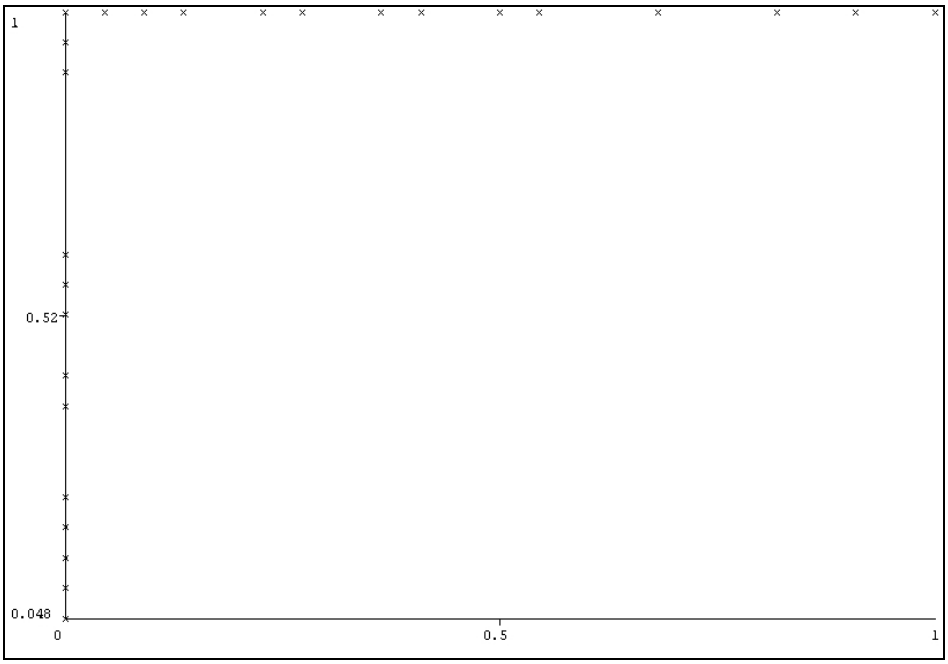


Figure 4.16 Real Attacks ROC Curve for Phd Help Desk with Naïve Bayes

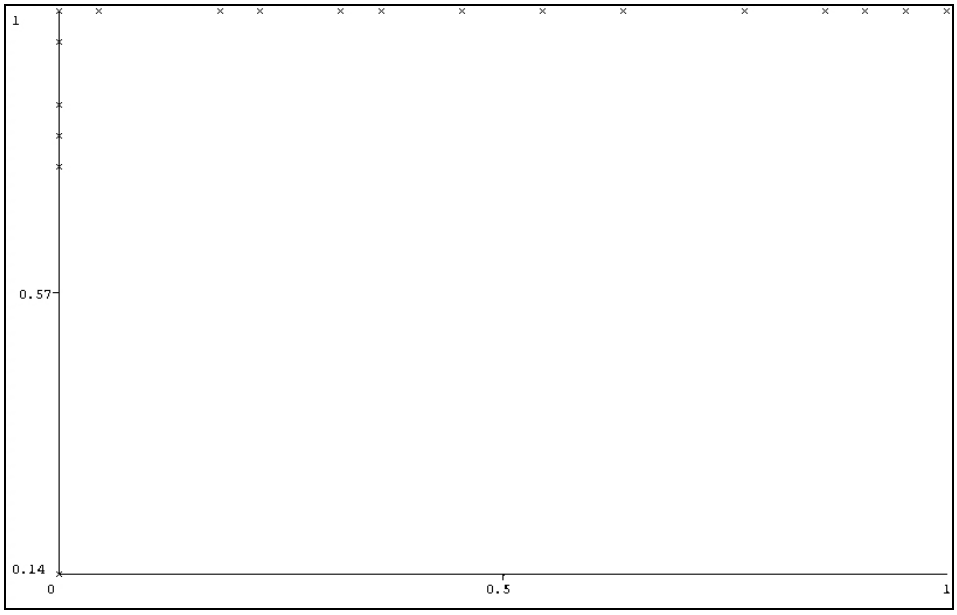


Figure 4.17 Real Attacks ROC Curve for Phd Help Desk with Random Forest

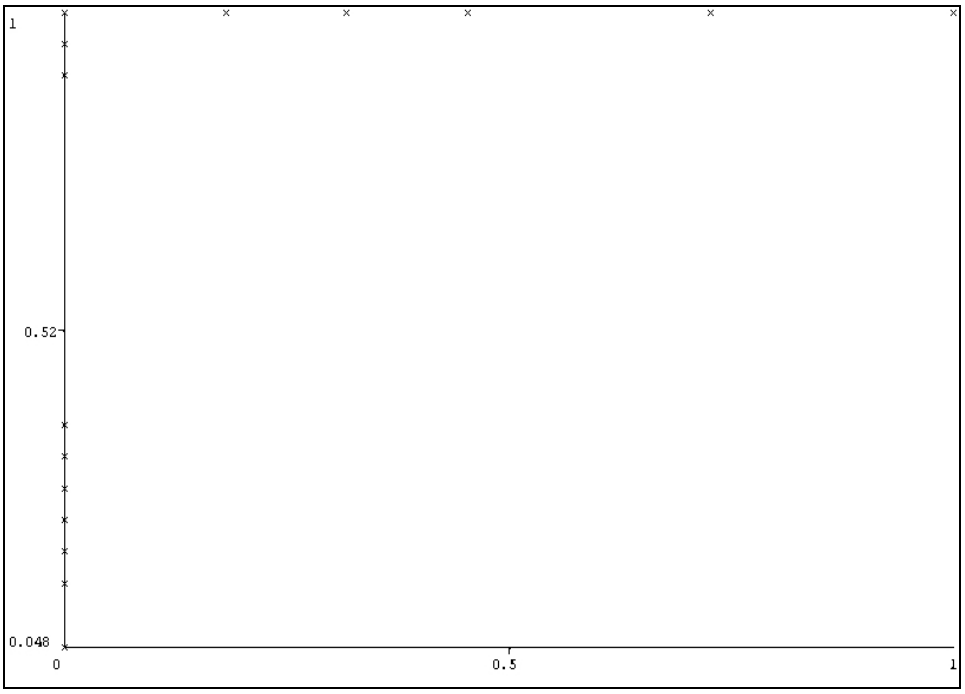


Figure 4.18 Real Attacks ROC Curve for Phd Help Desk with Rotation Forest

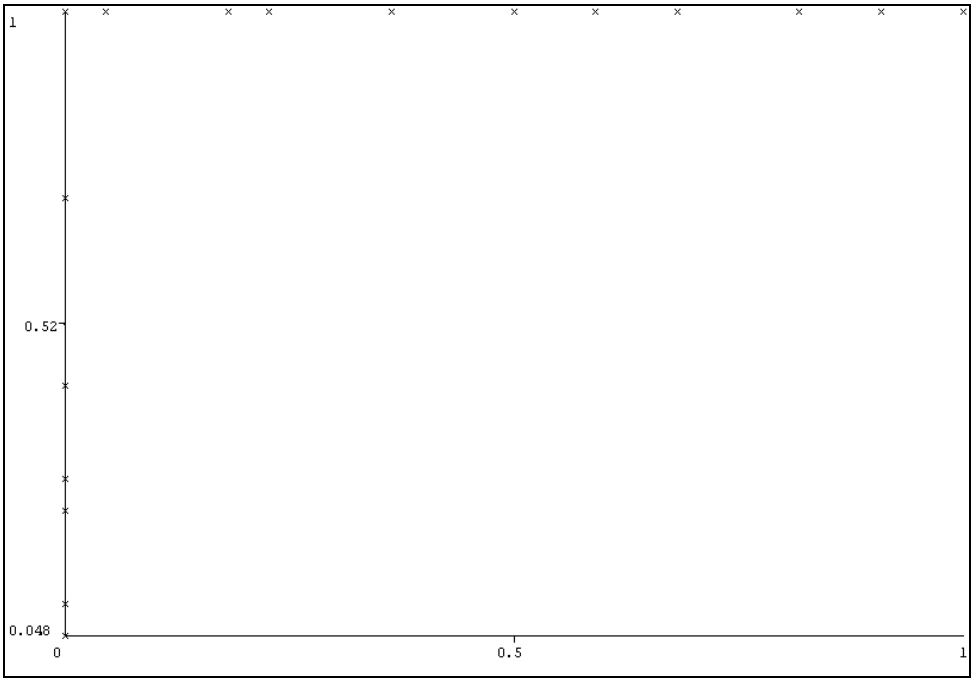


Figure 4.19 Real Attacks ROC Curve for Phd Help Desk with Simple Logistic

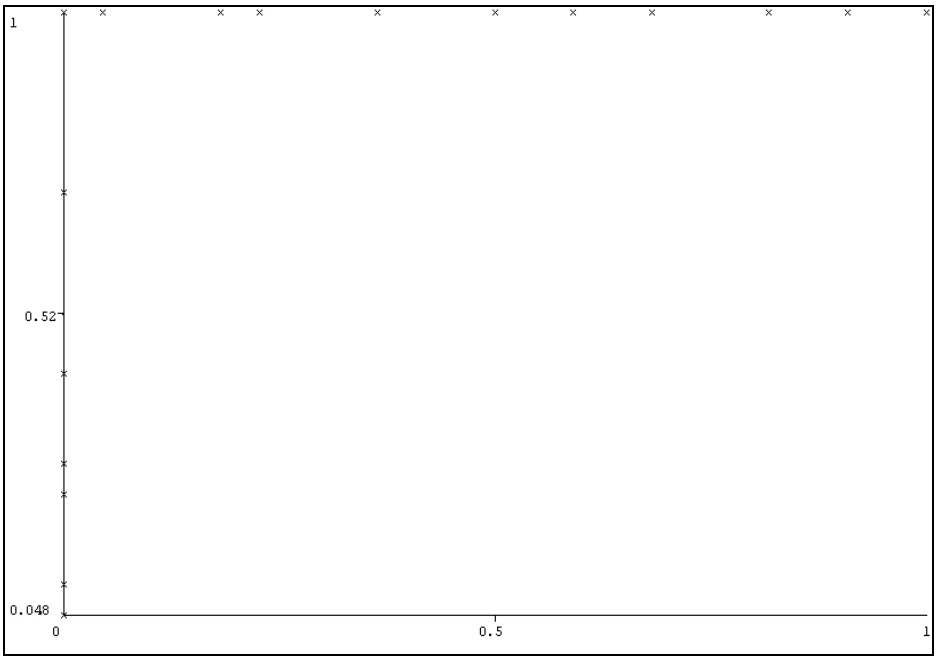


Figure 4.20 Real Attacks ROC Curve for Phd Help Desk with Aggregate Malicious

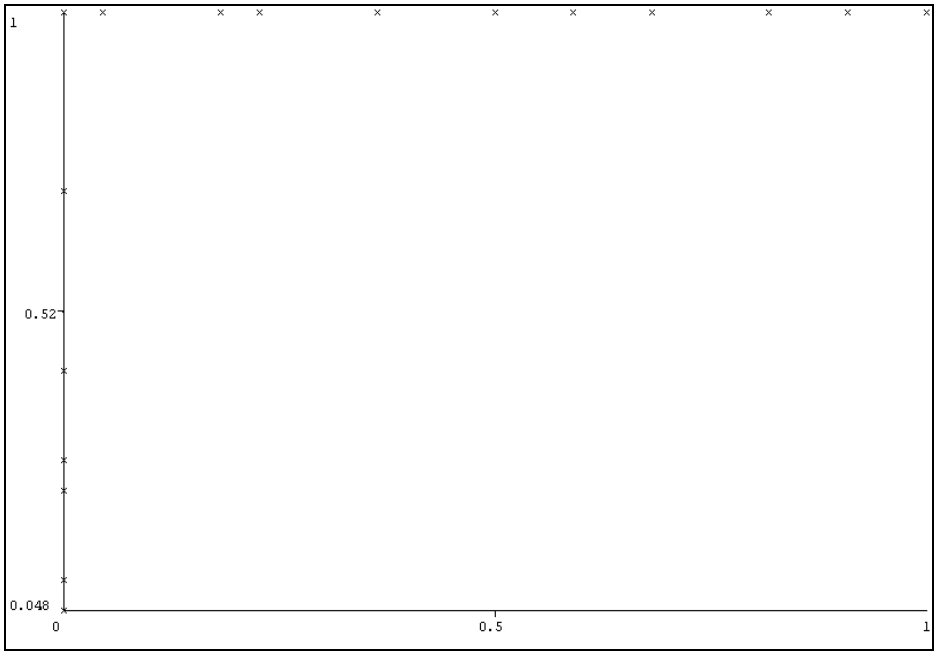


Figure 4.21 Real Attacks ROC Curve for Phd Help Desk with Aggregate Benign

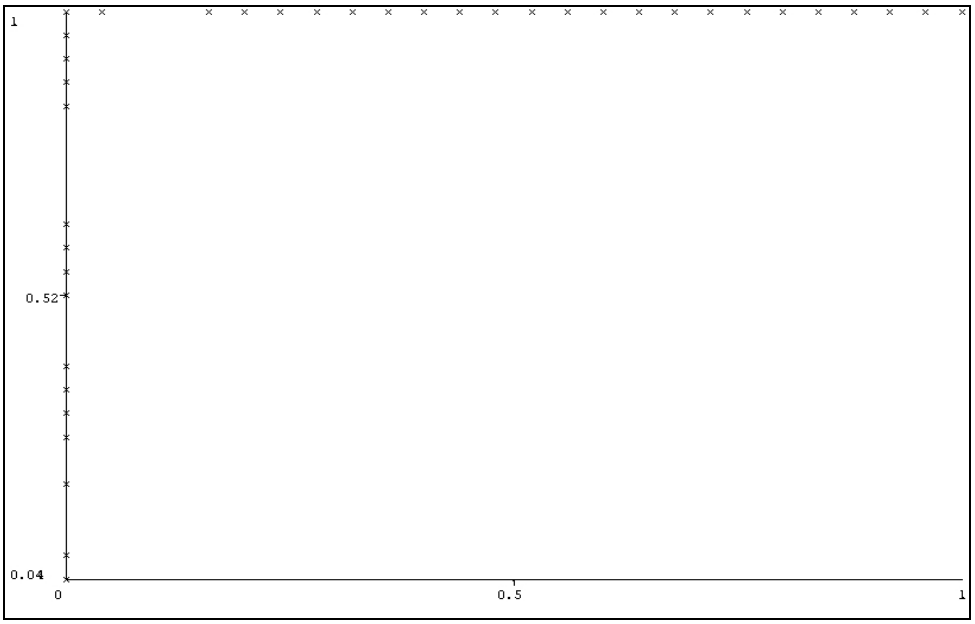


Figure 4.22 Real Attacks ROC Curve for OpenDocMan with Naïve Bayes

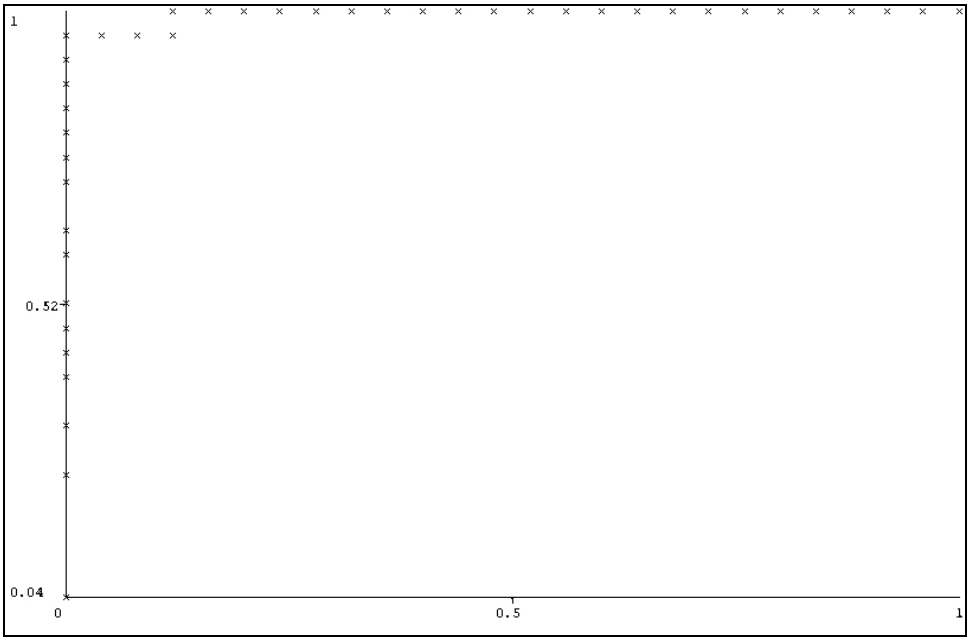


Figure 4.23 Real Attacks ROC Curve for OpenDocMan with Random Forest

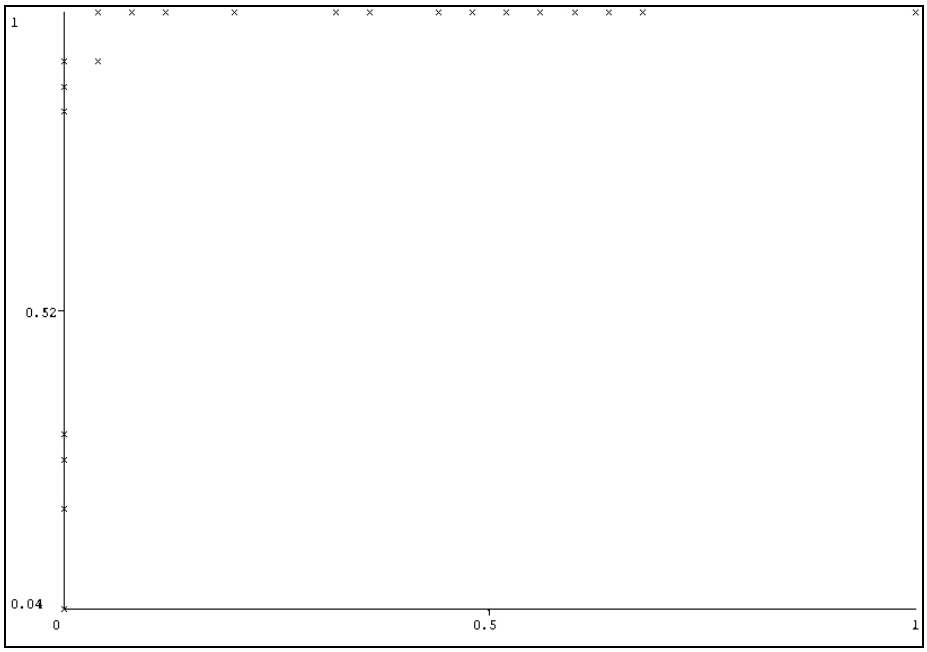


Figure 4.24 Real Attacks ROC Curve for OpenDocMan with Rotation Forest

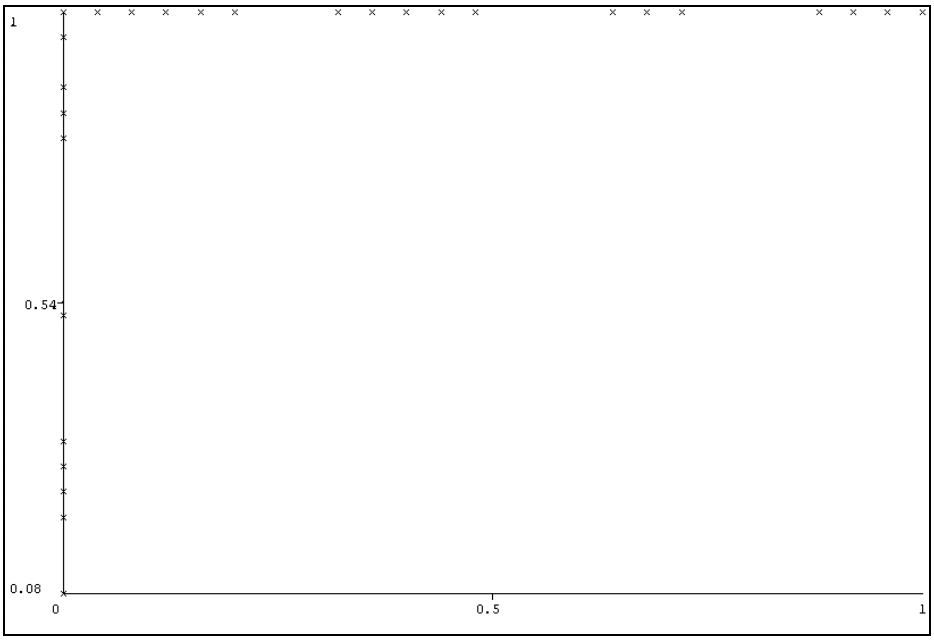


Figure 4.25 Real Attacks ROC Curve for OpenDocMan with Simple Logistic

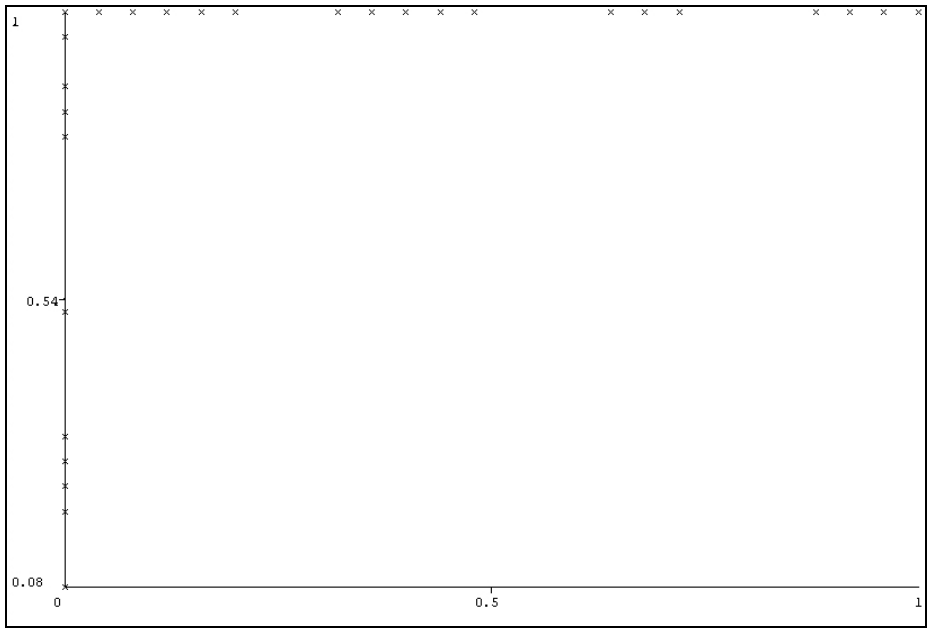


Figure 4.26 Real Attacks ROC Curve for OpenDocMan with Aggregate Malicious

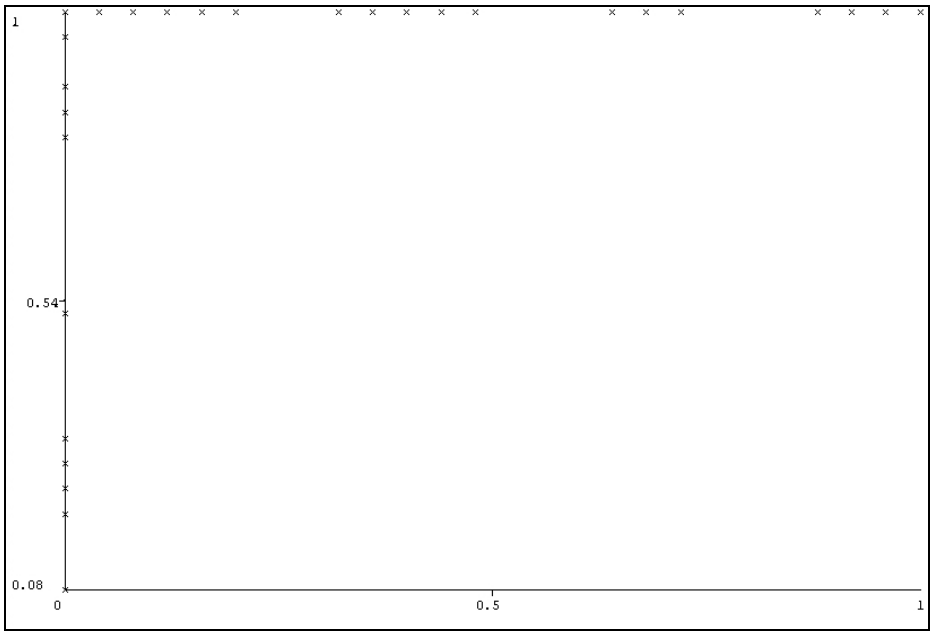


Figure 4.27 Real Attacks ROC Curve for OpenDocMan with Aggregate Benign

The results from the metrics for the "corner" of the curves are detailed in Table 4.3. This table shows that, similar to Section 4.2.1, the proposed IM is effective with all the ML algorithms. Furthermore, the aggregate algorithms can be seen to having no false positives and false negatives for both Phd Help Desk and OpenDocMan.

Table 4.3 Accuracy metrics

		Phd Help Desk	OpenDocMan
Naïve Bayes	Precision	1	0.903
	Recall	1	0.880
	F-measure	1	0.878
	Kappa	1	0.760
	MCC	1	0.783
Random Forest	Precision	0.906	0.931
	Recall	0.884	0.920
	F-measure	0.882	0.919
	Kappa	0.769	0.840
	MCC	0.790	0.851
Rotation Forest	Precision	1	0.996
	Recall	1	0.996
	F-measure	1	0.996
	Kappa	1	0.920
	MCC	1	0.923
Simple Logistic	Precision	1	1
	Recall	1	1
	F-measure	1	1
	Kappa	1	1
	MCC	1	1
Aggregate Malicious	Precision	1	1
	Recall	1	1
	F-measure	1	1
	Kappa	1	1
	MCC	1	1
Aggregate Benign	Precision	1	1
	Recall	1	1
	F-measure	1	1
	Kappa	1	1
	MCC	1	1

Because the results from Table 4.3 show that the algorithms are quite effective, a degree of agreement test was performed for information about error. If the degree of agreement is low, either

1. the IM does not allow the ML algorithms to function properly; or
2. AIWAIS is highly dependent on a specific IM.

For this test, the Cohen index is used because it is defensible as both chance-corrected measures and intraclass correlation coefficients (Fleiss 1975). The three "most" effective algorithms (Simple Logistic, Aggregate Malicious, and Aggregate Benign) are tested against all other algorithms. The results are shown in Table 4.4. This table shows that the algorithms are highly agree-able with each

other; hence, the IM is effective, and AIWAIS is not highly dependent on a specific algorithm.

Table 4.4 Degree of agreement

	Phd Help Desk			OpenDocMan		
	Simple Logistic	Aggregate Malicious	Aggregate Benign	Simple Logistic	Aggregate Malicious	Aggregate Benign
Naïve Bayes	1	1	1	0.76	0.76	0.76
Random Forest	0.77	0.77	0.77	0.84	0.84	0.84
Rotation Forest	1	1	1	0.92	0.92	0.92
Simple Logistic	1	1	1	1	1	1
Aggregate Malicious	1	1	1	1	1	1
Aggregate Benign	1	1	1	1	1	1

Table 4.5 shows the metrics for the "corners" of the ROC curves for the six algorithms after being trained with SMOTE balanced training data sets. Surprisingly, some of the algorithms cannot detect the attacks as well as when they are trained with the unbalanced data set. However, the two aggregate algorithms can be seen to have little to no decrease in performance; furthermore, they perform better than the other four.

Table 4.5 Accuracy metrics with SMOTE

		Phd Help Desk	OpenDocMan
Naïve Bayes	Precision	1	0.838
	Recall	1	0.760
	F-measure	1	0.745
	Kappa	1	0.520
	MCC	1	0.593
Random Forest	Precision	0.853	0.946
	Recall	0.791	0.940
	F-measure	0.782	0.940
	Kappa	0.585	0.880
	MCC	0.643	0.886
Rotation Forest	Precision	1	0.941
	Recall	1	0.940
	F-measure	1	0.940
	Kappa	1	0.880
	MCC	1	0.881
Simple Logistic	Precision	1	0.838
	Recall	1	0.760
	F-measure	1	0.745
	Kappa	1	0.520
	MCC	1	0.593
Aggregate Malicious	Precision	1	1
	Recall	1	0.960
	F-measure	1	0.980
	Kappa	1	0.960
	MCC	1	0.961
Aggregate Benign	Precision	1	1
	Recall	1	1
	F-measure	1	1
	Kappa	1	1
	MCC	1	1

4.2.3 Discussion of the Results

The figures and ROC curves from Sections 4.2.1 and 4.2.2 show some important results. AIWAS is shown to be effective at identifying malicious IMs. Although WA1 cannot be evaluated with real attacks, the well accepted 10 fold-cross validation approach demonstrates that the IM allows the ML algorithms to classify the HTTP requests with a high degree of accuracy. Attacks based on real vulnerabilities are also shown to be identified effectively (Section 4.2.2) by AIWAS.

The results show that AIWAS can be effective regardless of which ML algorithm is used. However, the detection rate does vary somewhat depending on which

algorithm is used. Whether certain algorithms are better cannot be concluded in this case study. However, because of the Principal of Aggregation (Rushton et al. 1983) and the results from Table 4.5, system administrators should use either aggregate algorithm with AIWAS. The question on which form of aggregate should be used would depend upon the relative costs of misclassifying for any particular application.

This page is intentionally left blank.

Chapter 5 – Estimating Reliability from the Server Logs

Reliability is becoming increasingly important to web systems due to the popularity of web applications. The need for highly reliable systems will only grow as companies continue to move their operations online. In order to increase reliability, a method to measure current systems' reliability is required. However, existing methods to measure reliability (Lyu 1995, Musa et al. 1987, Trivedi 2001) cannot be applied directly to web systems due to their specific nature (Alagar and Ormandjieva 2002, Offutt 2002). Thus, these existing methods will need to be modified to include new workload characteristics to estimate the reliability of web systems (Tian et al. 2004). More specifically, they defined two special characteristics:

- Massiveness and diversity: Web systems can interact with many different external systems. For example, one application may interact with Internet Explorer 6.5 and MySQL 3.23; another application may interact with Internet Explorer 5.5, Mozilla FireFox 1.5, SQLite 3.4.2 and Google Maps API 2.1. Not only that, every user with an Internet connection is considered to be a potential user of the web system. The workload characteristics selected need to reflect this diverse software configuration and massive and ill-defined user population.
- Document and information focus: Traditional workload concentrates on the computational focus whereas web systems principally have a document and information focus. Newer web systems have increased computation; however, search and retrieval remains the dominant usage for web users. The workload types for computational focus are fundamentally different than the workload types for document and information focus.

To measure web workloads to ensure accurate reliability estimation, generic workloads suitable for traditional computation-intensive cannot be used. Hence, Tian et al. (2004) defined four different web workload characteristics for reliability calculations:

- **The number of hits**: This workload is popular because each hit corresponds to a specific request to a web server, and each entry in the access log is a hit which allows for easy extraction of the data. However, this workload is misleading if it shows high variability with the individual hits (Tian et al. 2004).
- **The number of bytes transferred** may be used as a workload of finer granularity than the hit count; the number of bytes of transferred for each hit is recorded in the server logs and can be extracted with relative ease.
- **The number of users**: This alternative workload can be used by organizations that support various web systems and want to examine reliability at the user level. To count the number of users per day, the total number of unique IP addresses for that day is counted, and each unique IP address is assumed to correspond to a unique user. In other words, all hits

originating from the same IP address (which may be associated with one computer or multiple computers sharing the same IP address) are considered to be requests from a single user. A disadvantage of the user workload is its coarse granularity. This problem can be remedied by counting the number of user sessions.

- **The number of sessions** can be calculated from the IP address and the access time. If the time between each hit from one IP is within a time period, then all of these hits are considered to be one session. The session workload is better than the user workload because each session is typically associated with a change in user activity or a change in user. The same user may have several different usage patterns for each session; this can be revealed by the session workload characteristic.

Given the issues related to these workload estimates, this study will also examine simply using "days" as a workload characteristic. A "day" is defined as a 24 hour period within a log file. Clearly this alternative has a substantially coarser granularity than the alternatives discussed above. While the most obvious temptation is to utilize a fine-grain workload metric, since issues exists in their estimation, the question of are they actually a superior choice of normalizing term needs to be considered.

Although web traffic characteristics have been explored in detail – such as the characterization of the workloads (Alagar and Ormandjieva 2002), traffic trends and patterns (Crovella and Bestavros 1997), response times (Cremonesi and Serazzi 2002), etc. – only a few studies have investigated web error behavior and the measurement of web reliability. Although several hypothetical approaches exist; they lack empirical validations (Alagar and Ormandjieva 2002, Wang and Tang 2003). One practical approach to measuring the reliability of web systems is to use the information contained in server logs (Huynh and Miller 2005, Kallepalli and Tian 2001, Tian et al. 2004), such as system usage and failure codes. This information can be extracted and used to evaluate the system's reliability and identify "areas" for reliability improvement.

In this chapter, the approach of measuring reliability from server logs, as presented by Tian et al. (2004), will be evaluated and analyzed to determine the viability and effectiveness of this approach. Results from the original study and from this study will be used in the analysis. Two websites were examined in the original study; and two additional websites will be investigated in this study. Initially, these two websites are analyzed using the same methodology as proposed in the original study (Tian et al. 2004). That is, the server logs from these two websites were parsed for all errors that occurred while the websites were serving content to their visitors. A reliability estimate is then calculated from the extracted errors. This chapter extends the original study (Tian et al. 2004) by:

- Applying the technique to two new websites. One of which is a commercial website; in fact, the site can be considered as being mission critical to the commercial organization. The logs investigated for this

commercial website cover a 15 month period, which is an extensive time period. It is believed that this log represents the longest period of capture, and the only truly "mission critical" log reported within the research literature.

- Examining the error codes more rigorously; this will allow web administrators to focus on high value error codes.
- Re-examining the workload models to provide alternative methods for web administrators to analyze and interpret reliability information.

The remaining sections of this chapter are organized as follows: Section 5.1 describes the research methodology. Section 5.2 provides a brief overview of the characteristics of the websites used in the previous and the current study. Section 5.3 examines the workloads, the limitations of the workloads proposed, and the results from the two websites.

5.1 Research Methodology

Tian et al. (2004) demonstrated by performing an experiment on two websites that the operational reliability of websites could be estimated from server logs. They identified three failure sources:

- Host, network, or browser failures that prevent the delivery of requested information to web users. These errors can be analyzed and assured by existing techniques (Lyu 1995, Musa et al. 1987, Trivedi 2001) because they are similar to failures in regular computer systems, network or software (Tian et al. 2004).
- Source content failures that prevent the acquisition of the requested information by web users because of problems such as missing or inaccessible files, trouble with starting JavaScript, etc. These failures have unique characteristics to web systems (Crovella and Bestavros 1997, Montgomery and Faloutsos 2001, Offutt 2002); hence, special workload characteristics need to be defined before their reliability can be estimated.
- User errors, such as improper usage, mistyped URLs, etc. These errors also include any external factors that are beyond the control of web service or content providers.

They noted that host, network, browser failures and user errors can either be addressed by existing approaches or are outside of the responsibility and control of the content provider. However, source content failures represent a significant part of the problem and the content providers can address these issues. Hence, Tian et al. (2004) focused on web source content failures contained in error and access log files in their study. These files are created by all commercial HTTP Daemons.

The Nelson model (Nelson 1978), a widely used input domain reliability model, was used by Tian et al. (2004) to calculate reliability after the necessary information was extracted from the server logs. The formula for the Nelson model is:

$$R = \frac{n - f}{n} = 1 - \frac{f}{n} = 1 - r \qquad (1)$$

where f is the total number of failures, n is the number of workload units and r is the failure rate. The mean time between failures (MTBF) was then calculated as:

$$MTBF = \frac{1}{f}\sum_i t_i \qquad (2)$$

where t_i is the usage time for each workload unit i. If the usage time is not available, the number of workload units is then used as an approximation of the time period. Thus, the MTBF can be calculated as:

$$MTBF = \frac{n}{f} \qquad (3)$$

5.1.1 Removal of Automated Requests

The log files contain requests from robots and other automated systems that should be removed as they are not actual requests from web users. Automated systems are classified as systems that repeatedly request a resource from the website after a set period of time. For example, upon investigation of Site A's server log, requests from two monitoring services are identified. The first service requests a resource from Site A every 30 minutes while the second service requests a resource from Site A every 66 minutes. The resources these services request are unique and not publicly available. Hence, removing them simply involves identifying these resources in the log files. Robots that automatically request the "robots.txt" resource are also removed from both Site A and ECE log files.

Although it is infeasible to remove all automated requests from the server logs, web administrators need to remove all identifiable requests. Several techniques to identify them can be used by web administrators to remove automated requests. Most well known robots have a signature line that is included with every request as part of the USER AGENT field of the log file. For example, "Googlebot-Image/1.0" can be used to identify a robot from Google that is indexing the website's images. For web monitoring services, web administrators can simply dedicate a special resource that only these services can access. This resource can then be easily identified within the log files.

5.1.2 Analysis of Error Code Information

Error response codes can be extracted from either access or error logs. Due to the lack of error log files for the KDE website and Site A, only the access log files were used to extract the error information (Tian et al. 2004). Error response codes are embedded in the access logs, and these codes can be mapped to the error entries in the error log, for example, a "file not found" error in the error log usually corresponds to a 404 error code in the access log. Hence as stated in Tian et al. (2004), using just the access logs is a reasonable method to gather error information unless detailed information about the errors is required. Figure 5.1 provides a sample entry that can be found within the access logs.

```
129.194.12.3 - - [03/Nov/2005:15:44:34 -0500] "POST /data/search.php
HTTP/1.0" 200 50482 "http://www.sitea.com/data/form.php " "Mozilla/4.0
(compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.1.4322)"
```

Figure 5.1. A sample entry in an access log

This figure shows that on November 3, 2005, a remote user with an IP address of 129.194.12.3 used the POST protocol to access a file called search.php. The server responded with a 200 code and returned 50482 bytes of data. The previous URL that the user visited is http://www.sitea.com/database/form.php. The user used Microsoft Internet Explorer version 6.0 to access the web page.

The Nelson model and MTBF calculation require that the server logs capture the entire workload for the period under investigation. To ensure that the logs are complete, the parser used was customized to report suspicious gaps, which can be defined as long periods of inactivity between two recorded hits. These gaps were manually examined and discussed with the web administrators to ensure that the gaps are naturally occurring and not due to external factors such as the hard drive being full.

The error response codes in Tables 5.3 – 5.5 are the standard HTTP error response codes as defined by the Request For Comment (RFC) 2616[40] as part of the HTTP protocol. The following is a list of the codes encountered, their descriptions, and what the implications are when they are used for reliability analysis:

- 400 (Bad request) – the request could not be understood by the server due to its malformed syntax. This code should not be used for reliability analysis because the code is caused by a client that is not following the HTTP standard. Since this is a client-side issue, it does not make sense to estimate a website's reliability based on this code.
- 401 (Unauthorized) – the server does not accept the client's authorization credentials (or they were not supplied). This error occurs when a user requests a resource that the user does not have permission to retrieve. If the referrer for this resource is external to the website then this error can be ignored because the web administrators cannot control these external referrers. However, if the referrer is internal to the website and it is not the expected behavior of the server, then this error needs to be included in the reliability analysis. This situation of an error response code encompassing error types which are source content failure and external sources (human and system errors) occurs repeatedly; hence, the situation needs to be resolved to provide accurate reliability information. This issue is resolved later in the paper.
- 403 (Forbidden) – the server is refusing to fulfill the client's request. The cause for this error is similar to the 401 error code. Depending on the configuration of the HTTP daemon, this error may be returned instead of

[40] http://www.w3.org/Protocols/rfc2616/rfc2616.html, last accessed February 6, 2010

the 401 error code. Hence, it has the same issue as the 401 error response code, and will be discussed later.

- 404 (Not found) – the server cannot find anything matching the Request-URI. This error is currently the dominating error code and represented the focus of result of Tian et al.'s paper (2004). However, again, this error response code covers a multitude of different error types some of which are source content failure but others lie outside the system or what seem to be source content failures are actually not source content failures. For example, an attacker utilizing a scanner can (Spitzner 2001) spoof the referrer field of the log file when scanning for a system's vulnerability; the spoofed referrer field appears to be an internal link when it is actually from an external source. Links to old versions of the website can also create 404 error codes that appear to be internal bad links because the old version of the website is hosted on the same server as the current website. However, these internal bad links should be discarded because the user is using an incorrect version of the website. With the availability of powerful link checkers (NetMechanic HTML Toolbox[41], W3C Link Checker[42]), it is highly likely that actual source content failures are on the decline.
- 405 (Method not allowed) – the method specified in the Request-Line is not allowed for the resource identified by the Request-URI. The client performs a request that is not allowed by the server. For example, the client tries to perform a PUT request, but the server is configured to not accept PUT requests; hence, a 405 error code is generated. Since this error code only occurs due to a configuration issue, it should be discarded.
- 406 (Not acceptable) – this error is returned if the web server detects that the client cannot accept the data it wants to return. This error code should be discarded because the server's content does not support the client used to access it.
- 407 (Proxy authentication required) – if the client does not authenticate itself with the proxy then this error is returned. This error code can be discarded because the client did not authenticate with the server before attempting to access restricted content.
- 408 (Request timeout) – the client did not produce a request within the time that the server was prepared to wait. This is a network failure rather than a source content failure, and hence, it should be discarded.
- 409 (Conflict) – the client is attempting to perform a request that conflicts with the server's established rule. For example, the client is attempting to upload a file that is older than the file currently available on the server, this results in a version control conflict. This error can be discarded because it is a browser failure, not a server failure.
- 410 (Gone) – the server cannot find the requested resource and no alternative location can be found. This error code is related to the 404

[41] http://www.netmechanic.com/products/maintain.shtml, last accessed February 6, 2010
[42] http://validator.w3.org/checklink, last accessed February 6, 2010

response code, and hence it should follow the same rules as the 404 response code.

- 411 (Length required) – the server is denying the data the client is uploading because the client is not specifying the size of the data. Because this error is a browser failure and not a server failure, it can be discarded.
- 412 (Precondition failed) – the resource requested failed to match the established preconditions. This error should be included because the server failed to satisfy the preconditions; this implies that this error response code is a server failure.
- 413 (Request entity too large) – the server is rejecting the data being uploaded from the client because the data size is too large. The size limit can be adjusted within the server configuration. Since this error code only occurs due to a configuration issue, it should be discarded.
- 414 (Request-URI Too Long) – the server returns this error code in the following situations:
    - The client (usually a browser) has converted values from a POST request to a GET request. The POST request can handle larger values than the GET request; thus, the error occurs when an extremely large POST request is converted to a GET request.
    - The client is attempting to exploit some type of vulnerability in the server. Usually, these exploits involve a large amount of malicious code being injected into the Request-URI. Some of these vulnerabilities include: buffer overflows (Cowan et al. 1998, Evans and Larochelle 2002, Wagner et al. 2000), SQL injections (Boyd and Keromytis 2004, Huang et al. 2003), cross-site scripting[43], etc.

  Generally, the first situation is rare, and hence it is usually safe to assume that a majority of 414 errors will correspond to attacks on the server or other users who are accessing the vulnerable website. Thus, by identifying these 414 errors, system administrators can identify attacks on their server system and take appropriate actions against the attackers. Although the 414 error code is useful to system administrators, it is not a source content failure and, hence, will be excluded from reliability analysis.
- 415 (Unsupported media type) – the server is refusing the request because the resource is in a different format from the requested format. For example, the browser requests a resource and specifies it as a text document; however, the server recognizes the requested resource as a binary file and not a text document. A 415 response code would be generated in this scenario. Since this error code is a browser failure and not a source content failure, it should be discarded.
- 416 (Requested range not satisfiable) – the client is requesting a file size's range that is invalid. This error occurs when the client, usually a download manager such as Getright (http://www.getright.com) or Wget

[43] http://www.cgisecurity.com/articles/xss-faq.shtml, last accessed May 15, 2008

(http://www.gnu.org/software/wget/wget.html), erred in its resume point calculation. Hence, this error code should not be used in reliability analysis.

- 500 (Internal error) – the server encountered an unexpected condition which prevented it from fulfilling the request. Bugs within various dynamic scripts running on the server cause this error code. Therefore, it must be included in any reliability calculation.
- 501 (Not implemented) – the server does not support the request type that the client is sending. For example, the browser tries to retrieve the header information of an ASP enabled web page, so it sends a HEAD request to the server. However, the server does not understand this request for ASP enabled web pages, so it returns 501 error response code. This error code should be included in reliability analysis.
- 502 (Bad gateway) – This error has two definitions depending on the HTTP daemon used. For Apache, this error occurs when the server, while acting as a gateway or proxy, received an invalid response from the upstream server it accessed in attempting to fulfill the request. Because this error response code only occurs when the Apache HTTP Daemon is acting in a different mode rather than actively serving web pages, this error should be discarded for servers using the Apache HTTP daemon. For IIS, Microsoft IIS' support page[44] describes this error as "You receive this error message when you try to run a CGI script that does not return a valid set of HTTP headers." In other words, this error code can be triggered by an error in the web application's output code. Thus, this error should be included in reliability analysis if the web software is running on the IIS platform.
- 503 (Service unavailable) – The server is overloaded and cannot serve further requests. For example, due to a popular marketing campaign for a website, many users decide to visit this site. The unexpected load caused by this sudden increase in traffic causes a major strain in the server's resources, which then leads to extremely slow response time or a server crash. For example, Toys R Us' website received a surge in traffic after it released its Big Book catalog. This surge in traffic overloaded the system's resources, which lead to an extremely slow response time. Numerous potential purchasers were turned away because of this slow response time[45].

  This failure response code is a host failure that can lead to extended availability issue if not resolved properly. Tian et al. (2004) stated that availability problems are generally perceived by web users as less serious than web software problems. They argued that users are more likely to be successful in accessing required information after temporary unavailability whereas software problems would persist unless the underlying causes are

[44] http://support.microsoft.com/default.aspx?scid=kb;en-us;318380, last accessed February 7, 2010.
[45] http://money.cnn.com/1999/11/19/technology/etail_tech/, last accessed May 15, 2008

identified and fixed. This argument is questionable because web users are much more impatient and less forgiving than traditional users, as discussed by many studies (Galletta et al. 2004, Masterson 1999, Nah 2002, Rose et al. 2001, Williams 2001). They typically move on to the next site if they encounter issues with the current site that they are browsing. From their perspective, if they cannot access the information they want then it is an error. Hence, although the 503 error response code corresponds to a host failure and not a source content failure, it must be included in reliability analysis.

- 504 (Gateway timeout) – this error only occurs when the server is acting as a gateway or proxy server, hence it should be discarded.
- 505 (HTTP version not supported) – the server does not support the HTTP protocol version used by the client. This error can be discarded because the client is not using the proper HTTP protocol version.

It should be noted that web systems can be configured to catch error codes and respond with a 200 OK code instead. While this strategy hides technical information from users, it does not allow the error codes to be logged properly if configured incorrectly. Hence, web administrators should ensure that error codes are still logged if this strategy is to be used.

5.2 Overview of the Websites

Tian et al. (2004) applied the proposed approach to two websites. The first website analyzed was www.seas.smu.edu, the official web site for the School of Engineering and Applied Science at Southern Methodist University (SME/SEAS). The log files contained data covering 26 consecutive days in 1999. The second website analyzed was www.kde.org (KDE). This is the official website for the KDE project. The overall traffic and user population for this website is significantly larger than the SMU/SEAS website. The logs contained 31 days of traffic data. During these 31 days, over 13 million hits were recorded. Both of these websites used the popular Apache HTTP Daemon (http://httpd.apache.org) to serve their web pages.

5.2.1 Overview of the Websites in This Chapter

This chapter re-analyzes the approach presented in the original study (Tian et al. 2004). It initially applies this approach to two new websites, and based on these results postulates an alternative approach. The first website is www.ece.ualberta.ca, the website for the Department of Electrical and Computer Engineering at the University of Alberta. This site – similar to SME/SEAS and KDE – although important to the organization, it is non-commercial and not mission critical. This website is a dynamic website that utilizes the ColdFusion (http://www.macromedia.com/software/coldfusion) scripting language, and the Apache HTTP Daemon (http://httpd.apache.org). To investigate the stability of the data, the log files were chosen to cover approximately 30 consecutive days in January 2005 (ECE1) and 30 consecutive days in March 2006 (ECE2). For the month of January, the ECE website received approximately 500,000 hits, 53,100 "unique" visitors and transferred a total amount of 4.8 Gbytes of data. During

March 2006, the ECE website handled 470,000 hits, 61,000 "unique" visitors and transferred a total amount of 6.2 Gbytes of data.

The second website is the website for a publishing company that specializes in online databases (Site A). This website differs from the previous websites in that it is very critical to Company A's operation and hence it needs to be extremely reliable. The website utilizes the PHP (http://www.php.net) scripting language, MySQL (http://www.mysql.com) for the backend database and is hosted on an Apache HTTP Daemon. In order to observe potential trends and patterns for this mission critical website, the log files chosen cover 15 months of operation from January 2005 to March 2006. This website's traffic is lower than the ECE website. However, it represents a typical business website. That is, the site is a dynamic website with a mixed amount of static and dynamic pages – these are pages generated dynamically depending on the customers' inputs; its users are customers who are either looking to purchase a product or to register for a training course. For the 15 months covered, Site A received approximately 1.9 million hits and 92,000 "unique" visitors. The site transferred 34 Gbytes of data. Table 5.1 displays the technologies used by, and reliability requirements for, the two websites under investigation. Unfortunately, the ECE site administrator only has an approximate reliability target for their installation. These two websites were selected for this investigation because they utilize similar web development technologies while having different reliability requirements. The two websites use a scripting language in addition to an HTTP daemon; with one of the sites (A) also using a DBMS for data management. Although the technologies used are similar, their reliability objectives are quite different. ECE – due to its non-mission critical nature – is expected to experience a few failures per month. Site A requires high reliability because the loss of customers and sales will occur if the site's failure occurs. In other words, Site A is expected to experience no more than one failure per month.

Table 5.1 Sites examined

Site	Technologies	Reliability Requirement
ECE	CodeFusion, Apache	A few failures per month
Site A	PHP, Apache, MySQL	No more than 1 failure per month

Table 5.2 provides a summary of the properties of the logs used in previous studies and this study. Websites with an asterisk are commercial websites.

Table 5.2 Comparison of data sets

		Log duration	Requests	Bytes Transferred
Goševa-Popstojanova et al. (2006a)	NASA-Pvt1	20 week	23 thousand	0.5 GB
	NASA-Pvt2	20 week	92 thousand	0.2 GB
	NASA-Pvt3	20 week	489 thousand	2.2 GB
	NASA-Pub1	20 week	93 thousand	9 GB
	NASA-Pub2	20 week	732 thousand	6.7 GB
	NASA-Pub3	20 week	108 thousand	4.6 GB
	CSEE	6 week	5.8 million	80.9 GB
	WVU	3 week	37.9 million	97 GB
	ClarkNet*	2 week	3.3 million	27.6 GB
	NASA-KSC	2 month	3.5 million	62.5 GB
	Saskatchewan	7 month	2.4 million	12.3 GB
Goševa-Popstojanova et al. (2006b)	WVU	1 week	15.8 million	34.5 GB
	ClarkNet*	1 week	1.7 million	13.8 GB
	CSEE	1 week	397 thousand	10.1 GB
	NASA-Pub2	1 week	39 thousand	0.3 GB
Tian et al. (2004)	SMU/SEAS	26 day	763 thousand	7.8 GB
	KDE	31 day	14 million	110 GB
This study	Site A*	15 month	1.9 million	34 GB
	ECE1	1 month	500 thousand	4.8 GB
	ECE2	1 month	470 thousand	6.2 GB

This table shows that the longest period that previous studies have collected data is over a 7 month period, compared to 15 months in this study. Furthermore, studies that use logs from commercial websites cover extremely short periods (1 to 2 weeks). This study investigates the log file from a commercial website for a much longer period (15 months). Hence, it is believed that this study presents the first long-term analysis of a (mission-critical) commercial website.

5.3 Results and Discussions

This section presents the results for the four websites, and discusses various issues encountered during this experiment and explains the similarity and differences between the original study and this study.

5.3.1 Results from the Original Study

Tian et al. (2004) discovered many issues associated with the extraction of workload data for reliability estimation. However, the log files provide information that allows available data for the hit count, byte count and user count to be extracted with ease. The session count can be derived based on a timeout value which can provide more granularity than the user count.

They found that the four proposed workload characteristics allow reliability assessments from different perspectives. Hence, systems administrators can choose the best workload characteristic depending on the situation. For example, administrators concerned with data traffic measurement can examine the byte count whereas the hit count can provide more useful information regarding web users. The next section will present results found in this study and whether they confirm findings from Tian et al. (2004) study.

5.3.2 Results from this Study

Tables 5.3 – 5.5 provide a summary of the error response codes for all four websites. These tables contain the actual number of error counts and their corresponding percentages; these tables follow the analysis performed by Tian et al. (2004). That is, the access logs are parsed, and the errors are grouped together according to the error code without explicit considering of their cause. The original study provided only limited information for the KDE website; hence all the cells containing "n/a" are missing information that cannot be derived. Furthermore, the total percentage of errors recorded does not equal to 100 percent for this website. While Goseva-Popstojanova et al. (2006a, 2006b) also performed analysis on the error codes, the results are combined into groups such as 4xx (all 400 level error codes) and 5xx (all 500 level error codes). Hence, results from Goseva-Popstojanova et al. (2006a, 2006b) cannot be included in these tables.

Table 5.3 Recorded errors

Sites	Error code			
	400	**401**	**403**	**404**
SMU/SEAS	2 (0.02%)	14 (0.046%)	2,085 (6.78%)	28,659 (93.17%)
KDE	n/a	n/a	n/a	785,211 (98.90%)
ECE1	202 (0.15%)	6 (0.00%)	44 (0.03%)	136,143 (99.81%)
ECE2	52 (0.05%)	4 (0.00%)	211 (0.19%)	112,751 (99.74%)
Site A (Jan05)	1 (0.06%)	3 (0.17%)	188 (10.90%))	1,500 (86.96%)
Site A (Feb05)	0	10 (0.53%)	162 (8.50%)	1,722 (90.44%)
Site A (Mar05)	1 (0.05%)	28 (1.29%)	194 (8.90%)	1,938 (88.94%)
Site A (Apr05)	2 (0.09%)	17 (0.72%)	190 (8.07%)	2,121 (90.06%)
Site A (May05)	4 (0.20%)	27 (1.33%)	130 (6.39%)	1,849 (90.86%)
Site A (Jun05)	1 (0.05%)	36 (1.65%)	213 (9.78%)	1,920 (88.11%)
Site A (Jul05)	0	36 (1.53%)	146 (6.19%)	2,158 (91.44%)
Site A (Aug05)	0	28 (1.04%)	194 (7.20%)	2,448 (90.87%)
Site A (Sep05)	0	13 (0.59%)	167 (7.54%)	2,018 (91.15%)
Site A (Oct05)	0	12 (0.46%)	159 (6.03%)	2,434 (92.30%)
Site A (Nov05)	0	19 (0.68%)	214 (7.69%)	2,525 (90.76%)
Site A (Dec05)	1 (0.04%)	13 (0.54%)	156 (6.43%)	2,223 (91.56%)
Site A (Jan06)	0	19 (0.58%)	231 (7.04%)	2,758 (84.11%)
Site A (Feb06)	0	19 (6.66%)	164 (5.66%)	2,602 (89.82%)
Site A (Mar06)	0	22 (0.61%)	259 (7.12%)	3,321 (91.31%)
Site A (Total)	10 (0.03%)	302 (0.81%)	2767 (7.40%)	33,537 (89.69%)

Table 5.4 Recorded errors (cont.)

Sites	Error code				
	405	**408**	**414**	**415**	**416**
SMU/SEAS	0	0	0	0	0
KDE	n/a	6,225 (0.78%)	n/a	n/a	n/a
ECE1	0	0	0	0	6 (0.00%)
ECE2	2 (0.00%)	1 (0.00%)	0	0	14 (0.01%)
Site A (Jan05)	1 (0.06%)	0	0	30 (1.74%)	2 (0.12%)
Site A (Feb05)	0	0	0	10 (0.53%)	0
Site A (Mar05)	0	0	0	17 (0.78%)	1 (0.05%)
Site A (Apr05)	0	0	0	25 (1.06%)	0
Site A (May05)	2 (0.10%)	0	0	17 (0.84%)	0
Site A (Jun05)	0	0	0	9 (0.41%)	0
Site A (Jul05)	0	0	0	20 (0.85%)	0
Site A (Aug05)	0	0	0	24 (0.89%)	0
Site A (Sep05)	0	0	0	16 (0.72%)	0
Site A (Oct05)	0	0	0	32 (1.21%)	0
Site A (Nov05)	0	0	0	24 (0.86%)	0
Site A (Dec05)	98 (4.04%)	0	0	26 (1.07%)	0
Site A (Jan06)	254 (7.75%)	0	0	17 (0.52%)	0
Site A (Feb06)	83 (2.87%)	0	0	29 (1.00%)	0
Site A (Mar06)	5 (0.14%)	0	0	30 (0.83%)	0
Site A (Total)	443 (1.19%)	0	0	326 (0.87%)	0

Table 5.5 Recorded errors (cont.)

Sites	Error code			
	500	**501**	**502**	**503**
SMU/SEAS	0	0	0	0
KDE	n/a	n/a	n/a	n/a
ECE1	7 (0.01%)	0	0	0
ECE2	10 (0.01%)	0	0	0
Site A (Jan05)	0	0	0	0
Site A (Feb05)	0	0	0	0
Site A (Mar05)	0	0	0	0
Site A (Apr05)	0	0	0	0
Site A (May05)	0	0	0	6 (0.30%)
Site A (Jun05)	0	0	0	0
Site A (Jul05)	0	0	0	0
Site A (Aug05)	0	0	0	0
Site A (Sep05)	0	0	0	0
Site A (Oct05)	0	0	0	0
Site A (Nov05)	0	0	0	0
Site A (Dec05)	0	0	0	0
Site A (Jan06)	0	0	0	0
Site A (Feb06)	0	0	0	0
Site A (Mar06)	0	0	0	0
Site A (Total)	0	0	0	6 (0.02%)

These tables show that the 404 error type dominates, as noted by Tian et al. (2004). They discovered that, for SMU/SEAS, 99.9 percent of the errors encountered were of types 403 and 404 with 404 errors accounting for 93.1 percent of the recorded errors. For KDE, 98.9 percent of the recorded errors were of type 404. According to the survey results from 1994 to 1998 by the Graphics, Visualization, and Usability Center of Georgia Institute of Technology (http://www.gvu.gatech.edu/user_surveys/), 404 errors are the most common errors that users encounter while browsing the web. Ma and Tian (2003) found that a majority of these 404 errors are caused by internal bad links (IBL) while only a small percentage are caused by external factors such as the user mistyping the URL, robots from search engines, external links (links from other websites), old bookmarks, etc. Tian et al. (2004) discovered that only 8.7% of the 404 errors encountered were caused by external factors for SMU/SEAS. Despite this conclusion, they did not provide convincing evidence that the majority of the recorded errors are in fact from source content failures. Furthermore, these tables shows that, although the 404 error type dominates, other error response codes also exist; and while the 404 error type may dominate numerically, no analysis exists as to the "value" (of the information) encoded within the various error types for web site administrators. Therefore, all of the error codes encountered will be examined to determine which errors are truly source content failures (have value) and which are attributed to other uncontrollable factors (no value). For example, the 404 response errors have no value for Site A because all of the 404 recorded errors are caused by factors outside of the site administrator's control whereas the 503 error response code is high in value – the site administrator is expected to respond and correct the 503 errors immediately due to the potential loss in sales and customers that this error code can cause.

One common argument is that if information is available, external failures can also be resolved. This argument is not valid for several reasons. A site administrator can only be reactive to external failures rather than being proactive. That is, until an external failure occurs, a site administrator will not have enough information to resolve that failure. Furthermore, depending on circumstances, the failure may not be resolvable. For example, an external website has a link to a web page on the web system under examination. However, due to recent changes, that web page is no longer valid. The site administrator will not be aware of this issue until a user follows the link from the external website. Once the failure occurs, the site administrator can attempt to resolve it by attempting to contact the external website's Webmaster to get the link updated. However, this process requires cooperation from the external website's Webmaster. Furthermore, the process becomes tedious when there are thousands of websites linking to this invalid web page. The site administrator can also attempt to redirect the user to the correct page. However, this requires the site administrator to have a complete mapping of all invalid pages to valid pages which is clearly infeasible. Because of these potential issues, the site administrator cannot resolve external failures adequately.

Based on the information above, the error response codes can be associated to one or more failure sources. Table 5.6 displays this association for the error codes discussed. Error codes that do not have an association with a source content failure or host failure will not be investigated because they are beyond content providers', or website administrators', control.

Table 5.6 Failure sources for the error codes

Error code	Host	Source content	Network or browser	User and external
400				✓
401		✓		✓
403		✓		✓
404		✓		✓
405				✓
408			✓	
415			✓	
416			✓	✓
500		✓		
501		✓		
502		✓ (IIS)	✓ (Apache)	
503	✓			

Table 5.6 shows seven error codes, 401, 403, 404, 500, 501, 502 (IIS), and 503 that have either source content failure (SCF) or host failure as a potential failure source; hence, these seven error codes will be examined in detailed in order to determine their exact failure sources. Further, the 401, 403 and 404 error codes have both source content failure and external failures as failure modes or sources. After intensively investigating the log files for the two web sites under study (Site A and ECE), it is discovered that, for these web sites, the source content failures can be classified into two types:

- SCF1 – these are errors on the website that should be identified and corrected by the site administrators or content providers. These errors can be identified by close examination of the referrer field:

  If the referrer field of an error contains the website's URL, then the error belongs to the SCF1 category.

- SCF2 – these are usually links from external websites pointing to an old version of the website under investigation. This old version still exists on the HTTP Daemon for archival purposes and has no connections to the current website. Hence, it is not maintained and can contain many bad links. When a user visits this old version – through search engines, old bookmarks, old emails, etc. – and clicks on one of these bad links, the log data will record that the error is caused by an internal source. Since, these errors are under the direct control of system administrators, they are

classified as source content rather than external failures. However, an argument can be made that they are of lower value than SCF1 type errors. For example, for the ECE site, these errors are considered by the site administrator as a "non-issue"; and a case can be made for either including them or excluding them from reliability calculations. Errors belonging to the SCF2 type can be identified using the following method:

For each error, the referrer URL should be noted and visited. If the URL leads to an old version of the website, then the error is of SCF2 type.

External failure sources – which account for the majority of the failures – can also be classified into two categories:

- ES1 – which are old links from external websites, search engines, old bookmarks, etc. These external links can be detected based on the referrer information - each entry in the log files contains a referrer field which provides the web page that links to the content the user is requesting:

  All 401, 403 and 404 errors having URLs – not from the same domain as the website – or the character "-" in the referrer field are of the ES1 type.

- ES2 – which are scanners being executed by attackers looking for known vulnerabilities contained in various web applications. These scanners can send spoofed information to the web server. The web server will generate internal 401 or 403 errors if the web administrators have set up security permissions for these applications, or internal 404 errors if the website does not use these web applications. ES2 errors can be identified by close examination of the errors:

  If the requested resources belong to web applications not installed for the website, then the errors are of ES2 type.

401, 403 and 404 errors belonging to the ES1 and ES2 types should be detected and discarded during the data analysis phase. Tables 5.7 and 5.8 display the percentages of the different failure categories for the 401, 403 and 404 error codes, respectively. Due to unavailable information, the errors from the original study cannot be classified into the types discussed. These tables show that ECE (1 and 2) and Site A have extremely low (less than 0.5%) or no 401, 403, and 404 error codes as source content failures. All 500, 501, and 502 error codes were discovered to be source content failures, which is expected because of the associations shown in Table 5.6.

Table 5.7 Possible error codes for reliability analysis

Sites	Error code							
	401				403			
	SCF1	SCF2	ES1	ES2	SCF1	SCF2	ES1	ES2
ECE1	0	0	6 (100%)	0	0	0	38 (86.36%)	6 (13.64%)
ECE2	0	0	4 (100%)	0	0	1 (0.47%)	164 (77.73%)	46 (21.80%)
Site A (Jan05)	0	0	3 (100%)	0	0	0	186 (98.94%)	2 (1.06%)
Site A (Feb05)	0	0	4 (40.00%)	6 (60.00%)	0	0	158 (97.53%)	4 (2.47%)
Site A (Mar05)	0	0	28 (100%)	0	0	0	193 (99.48%)	1 (0.52%)
Site A (Apr05)	0	0	17 (100%)	0	0	0	189 (99.47%)	1 (0.53%)
Site A (May05)	0	0	27 (100%)	0	0	0	130 (100%)	0
Site A (Jun05)	0	0	36 (100%)	0	0	0	213 (100%)	0
Site A (Jul05)	0	0	33 (91.67%)	3 (8.33%)	0	0	146 (100%)	0
Site A (Aug05)	0	0	25 (89.29%)	3 (10.71%)	0	0	193 (99.48%)	1 (0.52%)
Site A (Sep05)	0	0	13 (100%)	0	0	0	167 (100%)	0
Site A (Oct05)	0	0	12 (100%)	0	0	0	159 (100%)	0
Site A (Nov05)	0	0	19 (100%)	0	0	0	214 (100%)	0
Site A (Dec05)	0	0	13 (100%)	0	0	0	153 (98.08%)	3 (1.92%)
Site A (Jan06)	0	0	19 (100%)	0	0	0	230 (99.57%)	1 (0.43%)
Site A (Feb06)	0	0	19 (100%)	0	0	0	163 (99.39%)	1 (0.61%)
Site A (Mar06)	0	0	22 (100%)	0	0	0	239 (92.28%)	20 (7.72%)
Site A (Total)	0	0	290 (96.03%)	12 (3.97%)	0	0	2733 (98.77%)	34 (1.23%)

Table 5.8 Possible error codes for reliability analysis (cont.)

Sites	404 error code			
	SCF1	SCF2	ES1	ES2
ECE1	0	16 (0.01%)	135,950 (99.86%)	177 (0.13)
ECE2	0	10 (0.01%)	112,643 (99.90%)	98 (0.09%)
Site A (Jan05)	0	0	1,479 (98.60%)	21 (1.40%)
Site A (Feb05)	0	0	1,683 (97.74%)	39 (2.26%)
Site A (Mar05)	0	0	1,881 (97.06%)	39 (2.94%)
Site A (Apr05)	0	0	2,075 (97.83%)	46 (2.17%)
Site A (May05)	0	0	1,814 (98.11%)	35 (1.89%)
Site A (Jun05)	0	0	1,877 (97.76%)	43 (2.24%)
Site A (Jul05)	0	0	2,087 (96.71%)	71 (3.29%)
Site A (Aug05)	0	0	2,377 (97.10%)	71 (2.90%)
Site A (Sep05)	0	0	1,986 (98.41%)	32 (1.59%)
Site A (Oct05)	0	0	2,391 (98.23%)	43 (1.77%)
Site A (Nov05)	0	0	2,477 (98.10%)	48 (1.90%)
Site A (Dec05)	0	0	2,139 (96.22%)	84 (3.78%)
Site A (Jan06)	0	0	2,686 (97.39%)	72 (2.61%)
Site A (Feb06)	0	0	2,344 (90.08%)	258 (9.92)
Site A (Mar06)	0	0	2,983 (89.82%)	338 (10.18%)
Site A (Total)	0	0	32,279 (96.25%)	1,258 (3.75%)

Finally, Tables 5.9 and 5.10 display the error codes generated from source content and host failures that will be used for reliability analysis in this study. This table contains the 500, 501, 502, and 503 error codes in addition to a subset of the error response codes from Tables 5.7 and 5.8. The 401 error code is not included in

this table because they do not contain any source content failures as shown in Table 5.7. Tables 5.9 and 5.10 effectively demonstrate the low number of "errors" of interest, or value, experienced by live web sites (ECE and Site A). These numbers have significant implications of reliability analysis and models for these types of systems.

Table 5.9 Error codes to be used for reliability analysis

Sites	Error codes		
	403	404	500
ECE1	0	16 (69.565%)	7 (30.435%)
ECE2	1 (4.762%)	10 (47.619%)	10 (47.619%)
Site A (Jan05)	0	0	0
Site A (Feb05)	0	0	0
Site A (Mar05)	0	0	0
Site A (Apr05)	0	0	0
Site A (May05)	0	0	0
Site A (Jun05)	0	0	0
Site A (Jul05)	0	0	0
Site A (Aug05)	0	0	0
Site A (Sep05)	0	0	0
Site A (Oct05)	0	0	0
Site A (Nov05)	0	0	0
Site A (Dec05)	0	0	0
Site A (Jan06)	0	0	0
Site A (Feb06)	0	0	0
Site A (Mar06)	0	0	0
Site A (Total)	0	0	0

Table 5.10 Error codes to be used for reliability analysis (cont.)

Sites	Error codes		
	501	502	503
ECE1	0	0	0
ECE2	0	0	0
Site A (Jan05)	0	0	0
Site A (Feb05)	0	0	0
Site A (Mar05)	0	0	0
Site A (Apr05)	0	0	0
Site A (May05)	0	0	6 (100%)
Site A (Jun05)	0	0	0
Site A (Jul05)	0	0	0
Site A (Aug05)	0	0	0
Site A (Sep05)	0	0	0
Site A (Oct05)	0	0	0
Site A (Nov05)	0	0	0
Site A (Dec05)	0	0	0
Site A (Jan06)	0	0	0
Site A (Feb06)	0	0	0
Site A (Mar06)	0	0	0
Site A (Total)	0	0	6 (100%)

This section discussed various different error codes and how they may or may not contribute to reliability analysis. Care has to be taken when dealing with these error codes as they do contain limitations that may affect the accuracy of a reliability estimate. The next section will discuss the workloads and any limitations they may have and how those limitations can further impact reliability analysis.

5.3.3 Workload Analysis and Discussions

Table 5.11 contains the workloads for the four workloads explored by Tian et al. (2004). Session count uses the standard two hours of inactivity to mark an end of a session (Montgomery and Faloutsos 2001), while "session count 2" uses 30 minute of inactivity period which was also used in many previous studies (Catledge and Pitkow 1995, Cooley et al. 1999, Fu et al. 1999, Goseva-Popstojanova et al. 2004, Goseva-Popstojanova et al. 2006a-b, Menasce et al. 2000a-b). This 30 minute figure is based on a mean value of 25.5 minutes (rounded up) determined by Catledge and Pitkow (1995). This figure is also believed to be commonly used in many commercial web applications (Huang et al. 2004). For example, Google Inc. uses the 30 minute timeout value for their Analytics web application[46].

Table 5.11 shows that when the timeout period is decreased, the session count increases. This behaviour is expected because a shorter timeout period means that some longer sessions will be split into multiple shorter sessions. Because the number of errors remains constant, the increased session count means the reliability estimation will increase. This effect can be seen in Tables 5.13 and 5.14. Hence, choosing the correct timeout period for the session count is important if an accurate estimation of reliability is to be obtained. This table shows that during the months of January to March 2006, there seems to be a steady increase in traffic for Site A; this "increase in traffic" is expected because there was a marketing campaign launched during this period to attract more users. However, the three available data points are not sufficient to numerically prove this conjecture.

[46]http://www.google.com/support/googleanalytics/bin/answer.py?hl=en&answer=55463 last accessed May 18, 2008

Table 5.11 Workloads

Sites	Workload					
	hit count	byte count	user count	session count	session count 2	days
ECE1	369617	4531 Mb	53208	60922	72502	30
ECE2	347413	5874 Mb	59727	71141	82761	30
Site A (Jan05)	120699	2191 Mb	5015	5336	6036	30
Site A (Feb05)	108219	1953 Mb	4982	5353	6017	28
Site A (Mar05)	135282	2474 Mb	6175	6633	7572	31
Site A (Apr05)	117785	2229 Mb	5800	6144	6961	30
Site A (May05)	113304	2110 Mb	5539	5926	6707	31
Site A (Jun05)	120784	2309 Mb	5902	6220	6940	30
Site A (Jul05)	105950	2060 Mb	5664	5980	6715	31
Site A (Aug05)	112997	2068 Mb	5935	6321	7094	31
Site A (Sep05)	111592	1980 Mb	5680	6055	6905	30
Site A (Oct05)	117256	2167 Mb	6258	6749	7666	31
Site A (Nov05)	122300	2178 Mb	6321	6784	7574	30
Site A (Dec05)	107702	2042 Mb	5948	6303	7296	31
Site A (Jan06)	148865	2726 Mb	7325	7792	8724	30
Site A (Feb06)	134334	2653 Mb	6830	7255	8094	28
Site A (Mar06)	161266	3147 Mb	8233	8771	10405	31
Site A (Total)	1838335	34287 Mb	91607	97622	110415	453

In order to determine if any correlation between the workload characteristics exists, Principal Component Analysis (Jolliffee 1986) was performed. Table 5.12 shows the results for Site A (Total), and Figure 5.2 shows the Scree plot. The plot shows that only one component has an Eigen value over 1 and all other components after Component 1 appear to level off. This suggests that only one component is of importance. Results for the other websites (ECE1 and ECE2) are a similar, but are omitted for brevity. These results show that all of the workload characteristics are highly correlated which suggests that any workload characteristic can be used for reliability estimation. However, website administrators should select the workload characteristic most suitable for their requirements.

Table 5.12 Correlation matrix

	hit count	byte count	user count	session count	session count 2
hit count	1	0.95	0.91	0.91	0.91
byte count	0.95	1	0.92	0.92	0.91
user count	0.91	0.92	1	0.998	0.98
session count	0.91	0.92	0.998	1	0.99
session count 2	0.91	0.91	0.98	0.99	1

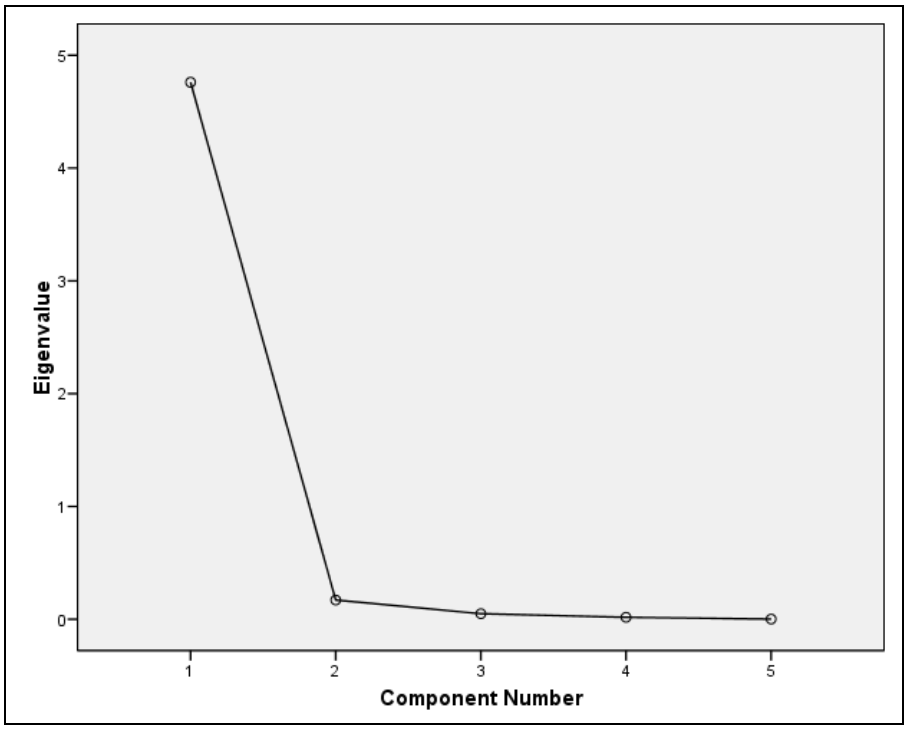


Figure 5.2 Scree plot

Tian et al. (2004) discussed the potential issues in using the byte count as a workload because a variety of entries, including error entries, in the access log that do not contain information on the number of bytes transferred. Upon further investigation, they discovered that the missing entries are associated with binary files already stored in the user cache. The byte count also treats large file size resources as more important than smaller sized resources. For example, let's assume that resources A and B exist on a web server, and resource A is much larger in size than resource B. A user, who requires both resources A and B, attempts to retrieve these two resources. Resource A failing will have a greater effect on the reliability estimation of the system, which is inappropriate because the reliability of the server is the same regardless of the size of the resource. Figure 5.3 shows the file size (in Kbytes) histogram for Site A which illustrates this issue. The figure shows that the size of the resources on the furthest right is equivalent to the combined size of many resources on the left side.

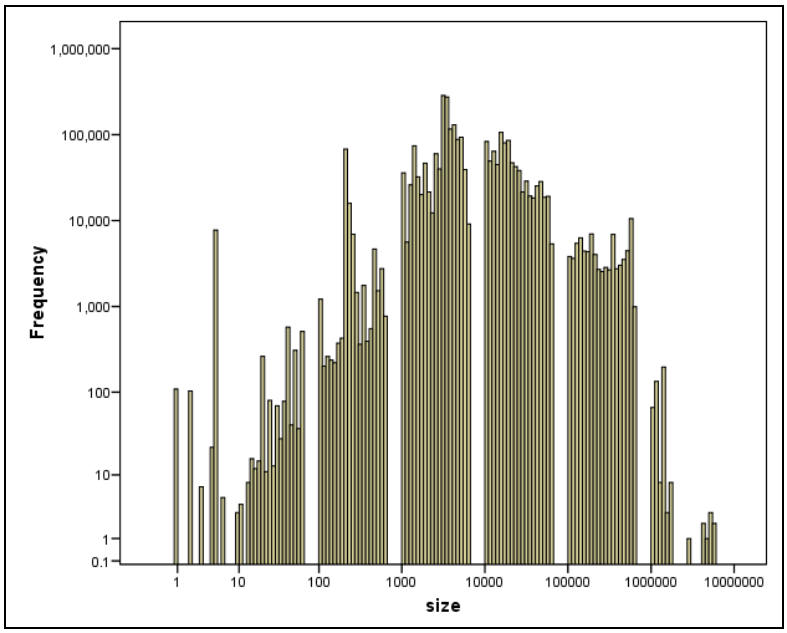


Figure 5.3 File size histogram for Site A

Other issues also exist with using the user count and session count as workloads (Alagar and Ormandjieva 2002, Arlitt and Jin 1999, Rosentein 2000). In fact, since web workload characterization was extensively examined by Arlitt and Williamson (1997), many studies have been performed to further examine the individual workloads (Arlitt and Jin 1999, Cherkasova and Phaal 1998, Menasce et al. 1999, 2000). Tian et al. (2004) suggested that each unique IP address can be counted as one user. However, with the current explosion in the number of Internet users, the total amount of IP addresses available is shrinking rapidly. Thus, many methods now exist that allow one public IP address to be used for a group of machines; some of these methods include proxy servers, and personal routers. Since the original study suggests counting one unique IP as a user, there is a strong possibility that this "user" is actually a group of users. As personal routers and proxy servers become more dominant this issue is also becoming more prominent. The session count also suffers this same problem because "one session" may actually be several sessions from several different users who are sharing the same public IP. Thus, a methodology needs to be developed to distinguish different users before accurate reliability analysis can be performed. Websites can use cookies to track user and sessions more effectively by placing a unique identifier and time related information inside the cookie. However, limitations still exist, such as two users sharing the same machine to access the website. The effectiveness of using cookies as a method to track user and session workloads will be explored in the future.

Results from this section confirm issues with the extraction of workload data from the server logs as discussed in the original study (Tian et al. 2004). Issues not discussed in previous studies (Tian et al. 2004, Goseva-Popstojanova et al. 2006a-b) such as file size bias and proxy servers, are also presented to ensure that web

administrators using this approach for reliability estimation are aware of these limitations.

5.3.4 Reliability Analysis and Discussions

The failures and workloads can be applied to the Nelson model to evaluate the overall operational reliability. Using equation (1), R, based on the hits workload, was calculated for the websites under examination; the results can be seen in Table 5.13. Not surprisingly, Site A, which has the highest reliability requirement, has a high reliability rate during the 15 month period (99.997% of the hits are successful). The sudden drop in reliability during May 2005 was examined; upon closer investigation and discussion with the administrator, it is discovered that a configuration setting was not set up correctly; hence the website experienced several simultaneous server failures.

Table 5.13 Reliability analysis

Sites	R
ECE1	0.999938
ECE2	0.999940
Site A (Except May05)	1
Site A (May05)	0.999947
Site A (Total)	0.999997

The hit reliability figures are consistent with previous studies (Tian et al. 2004, Goseva-Popstojanova et al. 2004, 2006a-b) in that they are very high. However, other workloads can be used to obtain different resolution for the reliability figure. As discussed by Tian et al. (2004) reliability based on other workloads (users, sessions, and bytes) can be calculated using:

$$R = 1 - \frac{f_w}{n_w} \qquad (4)$$

where f_w is the number of workloads with at least one failure recorded. For example, f_{users} is the number of users who encountered at least one failure. n_w is the total number of workload units. Goseva-Popstojanova et al. (2004, 2006a), using the Nelson model, discovered that reliability based on the session workload is lower than reliability based on the hit count. However, there is no straightforward relationship between hit reliability and session reliability (Goseva-Popstojanova et al. 2006a); hence, web administrators should not use these two metrics interchangeably. Table 5.14 displays reliability using the other workloads. This table shows all workload units provide extremely high reliability number due to the low error count associated with the websites under investigation. However, the "days" workload characteristic contains rates that are lower, especially for ECE (closer investigation revealed that the ECE website experienced a high failure rate per day which results in the low reliability figure). Hence, the advantage of the four workloads – being able to provide better granularity than the daily error rate – is lost. In addition, significant issues still exist in accurately estimating the four proposed workloads. Hence, any future work on "live" (as opposed to test) websites should simply utilize days as their

basis unless there are specific requirements that force web administrators to use other workload characteristics.

Table 5.14 Reliability analysis using the other workloads

Sites	Rbytes	Rusers	$R_{sessions}$	$R_{sessions2}$	R_{days}
ECE1	1	0.999565	0.999622	0.999683	0.233333
ECE2	1	0.999648	0.999705	0.999746	0.300000
Site A (Except May05)	1	1	1	1	1
Site A (May05)	1	0.998917	0.998988	0.999105	0.806452
Site A (Total)	1	0.999935	0.999939	0.999946	0.986755

The mean workload between failures (MWBF) can also be calculated using the model discussed in Section 5.1. This model may provide better estimation due to the fact that it does not have the same limitations that the Nelson model has. Furthermore, it allows web administrators to analyze failure based on time. The original study calculated the MWBF by substituting the number of workloads units for time, effectively using formula (3) for analysis; hence, this study also uses this formula to calculate the MWBF for the websites under investigation. The resulting MWBFs for the two websites can be seen in Table 5.15. Sites (or months) with "n/f" experience no failures during the time period measured. The MWBF data in Table 5.15 states that an error will be encountered for each of the workload (bytes, hits, users and sessions) values specified. This table shows that ECE1 has, on average, a failure for every 16,070 hits; Site A would experience one failure after every 306,389 hits. Looking at the "days" column shows that Site A does meet its reliability requirement of having no more than one failure per month (except in May), whereas ECE experiences at least one failure every week which is also expected.

Table 5.15 MWBF

Sites	hits	bytes	users	sessions	sessions2	days
ECE1	16,070	1.97×10^{08}	2,313	2,648	3,152	1.30
ECE2	16,543	2.80×10^{08}	2,844	3,387	3,941	1.43
Site A (Except May05)	n/f	n/f	n/f	n/f	n/f	n/f
Site A (May05)	18,884	3.52×10^{08}	923	987	1,117	5.17
Site A (Total)	306,389	5.71×10^{09}	15,267	16,270	18,402	75.5

The MWBF calculated using the second MTBF formula can only provide a rough estimate of the actual MTBF. Although using the workload units as a substitute for time is a reasonable method in situations where the time is not available, for this analysis, the time can be calculated from the daily failure. That is, MTBF (in hours) = 24×(daily failure rate).

ECE is an academic website; hence it is not surprising to see its MTBF to be at 31.2 hours (1.3 days) and 34.3 hours (1.43 days) as opposed to Site A which has a MTBF rate of 1,812 (75.5 days) hours for the entire 15 months. Again, the low

MTBF (relatively) rate for Site A during May 2005 can be attributed to the web application upgrade issue.

This section shows that reliability can be estimated from server logs and expressed in different metrics. Different reliability metrics have been examined to provide system administrators with the flexibility of selecting the correct metric based upon the requirements. For example, the requirements of Site A and ECE were expressed in terms of failures per month. Hence, system administrators for these websites can choose the MTBF to express their estimated reliability.

5.3.5 Limitation of Log Files

Although log files can provide failure information, reliability can only be estimated from them. The actual reliability cannot be computed solely from web servers' log files due to several issues. The workload information cannot be accurately computed as mentioned in Section 5.3.1. However, with the help of web technology such as cookies, developers are beginning to be able to track the user session count and user count more accurately. Techniques on identifying the correct timeout value for the session workload are also being discussed by various researchers (He and Goker 2000, Huntington et al. 2008). As these technologies and new techniques are being utilized, more accurate workload data will be gathered which will increase the accuracy of reliability estimation.

Furthermore, errors that are not recorded in the log files may lead to an inflated reliability figure. For example, a website's link may point to an incorrect web page rather than a missing one. This type of error requires human intervention as the error is only defined by a deviation from the specification rather than an exception. That is, the error codes in the server logs can only identify resource availability issues such as missing resources, moved resources, etc., and not whether the resources contain incorrect content. In this scenario, an error would not be recorded in the log files and the error would only be known when the customer reports the issue. Reliability estimation based on log files alone would not include this error. Because the link is available, automated web crawlers would not be able to detect this error. In fact, this scenario requires manual user intervention to detect the error; hence the error would have to be added manually to the data to increase the accuracy of the proposed reliability estimation method.

This page is intentionally left blank.

Chapter 6 – Empirical Observations on the Session Timeout Threshold

One of the most popular units used to analyze traffic, workload and user behaviour is the session. For example, Chen et al. (2003) presented several algorithms that allow web miners to efficiently calculate the number of user sessions with some session timeout threshold (STT). Pallis et al. (2005) proposed a technique to discover relationships between user sessions; the user sessions were identified using Chen et al. (2003)'s proposed technique. The session measure has also been investigated by many researchers (Goševa-Popstojanova et al. 2004, 2006a, 2006b; Arlitt and Williamson 1997, etc.). However, no model has been proposed to estimate the STT used to generate session length data.

Many other studies have also concentrated on determining various session-related workloads based on a predefined constant value for STT. For example, Montgomery and Faloutsos (2001) defined STT to be 2 hours long. Tian et al. (2004) used 15 minutes and 2 hours as two different STTs; these two STTs were then applied to both websites investigated in that study. Chen et al. (2003) and Goševa-Popstojanova et al. (2006a) assigned the STT value to be 30 minutes and use it for all websites in their studies. This 30 minute value is also used by other researchers (Berendt et al. 2001, Spiliopoulou et al. 2003, Mahoui and Cunningham 2000, Mat-Hassan and Levene 2005). Furthermore, this 30 minute figure is commonly used in many commercial web applications (Huang et al. 2004). For example, Google Inc. uses the 30 minute timeout value for their Analytics web application[47]. This figure is based on a mean value of 25.5 minutes (rounded up) determined by Catledge and Pitkow (1995). While this standard period is often used, it is far from obvious that it provides any meaningful guidance in estimating user session lengths. In fact, a recent study shows that session lengths can be as long as 6 hours and 32 minutes, the average period spent on RuneScape.com. Furthermore, with the advent of AJAX (Garrett 2005) and other interactive technologies, the session length values will be further impacted as websites' interactivity features begin to rival that of desktop applications. Hence, using the same STT for all websites may not lead to accurate results.

This chapter has the following contributions:

- A model, based on empirical observations, for estimating the STT is presented. Although the model has limitations, it provides an initial step that will allow future studies to expand upon. Furthermore, this model is proven to be applicable at many different resolutions and to two uniquely different websites.
- The concept that STT varies for each website is empirically proven. This encourages future research on web server logs to be performed using a

[47]http://www.google.com/support/googleanalytics/bin/answer.py?hl=en&answer=55463 last accessed August 21, 2008

customized STT value per website rather than a constant that's applied to all websites.
- Empirical investigation on data sets with very long collection periods. The benefits are discussed in Section 6.4.

The remaining sections of this chapter are organized as follows: Section 6.1 discusses current approaches used to identify STT. The new session threshold timeout model is proposed in Section 6.2. Section 6.4 provides a brief description of the websites under investigation, and the properties and characteristics of these websites. The results for this model when applied to the websites under investigation are discussed in Section 6.4.

6.2 Related Works

Other approaches have been proposed to calculate STT. For example, Catledge and Pitkow (1995) determined the STT to be 25.5 by claiming that the most statistically significant events occurred within 1.5 standard deviations (25.5 minutes) from the mean between each user interface event which was 9.3 minutes. However, no definition of "significant events" was supplied; and why 1.5 standard deviations is selected is never discussed. More importantly, only four percent of the accessed web pages were dynamic pages. Hence, the investigation was heavily based on static web content, which is increasingly rare in modern applications.

He and Goker (2000) performed an empirical investigation of the session value by initially setting STT to a very large value, then slowly decreasing it until they achieved a stable point where the number of activities remains stable for both short sessions and long sessions. However, the data used was very limited. Not only that, they provided no formal definition of the stable point of the system and provided a range of values of "somewhere between 10 – 15 minutes" for STT. They also claimed that this range is suitable for all websites on the World Wide Web. This finding is questionable because of niche specific websites that can attract different user demographic groups. For example, users visiting www.youtube.com can spend a long period online watching various video clips; while visitors of www.onlineconversions.com would use the site to perform quick metric conversions and quickly finish their sessions. Huang et al. (2004) proposed a dynamic approach to determining STT. Basically, the approach tries to detect general behavioural patterns of web-site usage. The approach assumes that these patterns can be approximated by sequences of hypertext interactions. A session "ends" when a user deviates from a learned pattern. However, an approach to determine the parameters used and how session identification results can be measured are not discussed. Not only that, the time for learning or discovering patterns is unknown and the site cannot be updated as the learning needs to be repeated whenever evolution takes place.

Huntington et al. (2008) proposed a set of STTs based on the content retrieved by the user. They demonstrated that the STT for each content type can be retrieved based on the estimated view time (the time between the logged request to

download the article and the next request) from the server log files. However, the method used to estimate the view time has several limitations. If the user requests a page, reads it, then closes the browser window without performing any additional action then the estimated view time would be inaccurate. Furthermore, most web pages contain multiple content types. For example, a web page can contain both a Menu content type and an Abstract content type; the authors do not discuss a method for classifying these pages and how the STT can be retrieved from these multiple content pages.

While the previous papers successfully introduced the idea of a session timeout threshold, their treatment of the concept was either exceptionally brief, imprecise or contained many unsolved issues. Given, the relative importance of this metric, it is believed that this situation needs to be urgently resolved. However, no simple unique definition of this concept is likely to exist; and hence a protracted investigation is required.

6.3 Observations of the STT and the Proposed Model

A session is defined as a sequence of actions undertaken by a user within a period of time. Sessions offer much finer grained information than the standard *number of users* metric. However, because the Hyper Text Transfer Protocol (HTTP) is a stateless protocol, session information cannot be easily captured. Hence, web applications often use session-based technology such as cookies (Kristol and Montulli 2000) to simulate a stateful connection to the user. In order to determine when a session ends and the next one begins, a session timeout threshold (STT) is often used. In other words, a STT is a pre-defined period of inactivity that allows web applications to determine when a new session occurs. That is, let t equal the session timeout threshold and s is the set of sessions:

$$\forall s \in \text{SessionsFor(user)} \bullet (\text{session\_time\_start}(s_{i+1}) - \text{session\_time\_end}(s_i)) \geq t$$

For web server logs, the STT is determined by the time between the current request and the previous request.

The user session metric is particularly interesting to web mining researchers because they provide finer grain of information than the usual user count. (Menascé et al. 1999), Arlitt and Williamson (1997) and Pitkow (1999) have noted that the number of sessions is directly dependent on t. Hence, it is important to select the correct t in order for the number of sessions to be estimated accurately.

A STT is best viewed as a "design parameter", a mechanism for website workload evaluation rather than a concept with an absolute definable theory. Hence, this discussion is best considered as an attempt to produce an initial model that will allow web administrators to estimate this design parameter for their websites. From a philosophical viewpoint, the definition of session timeout threshold has many of the characteristics of a "wicked problem" (Rittel and Webber, 1973).

That is, the problem has many complexities such as changing, incomplete or contradictory requirements. Hence, any solution will experience significant limitations. In fact, sessions and session timeout threshold are clearly ideas more in line with Simon's (1996) "sciences of the artificial" than "sciences of the natural". In this situation, practitioners seek "good enough" solutions rather than optimal solutions; or *satisficing* to utilize Simon's term from economics (Simon, 1955). This research recognizes these limitations.

Empirical observations of the number of sessions versus the STT show that the estimation of STT can be viewed as a partitioning problem. The problem can be approached as a question of defining two regions or surfaces (S_1 and S_2), which represent the lengths before and after the threshold value (x_i) respectively. In addition, the number of sessions obviously monotonically declines as STT increases. Within each region, the data points are "relatively stable and vary smoothly." S_1 represents a potentially steeply declining curve, where the choice of session length has a significant impact upon the result. S_2 is in fact two segments (S_2 and S_3); S_2 can be characterized as a linear segment with a "gentle gradient"; the choice in session length has limited impact in this region. Whereas S_3 is a second linear segment with no gradient; and can effectively be modeled by the constant number of sessions that it represents which is equal to the number of users and is the lower bound of the system. In terms of the model and its usage, the S_3 curve is unimportant and hence is not actively considered. The behavior across S_1 and S_2 can be modeled as a rate of change statement, or, more specifically, as a requirement to minimize the rate of change of the curvature across S_1 and the rate of change of the gradient across S_2.

Hence, the problem can simply be recast as a question of finding the threshold value that minimizes the overall rate of change of the system given that the system consists of two separate regions. Although this study also uses the minimal change as the STT as proposed by He and Goker (2000), a mathematical model, which is lacking in the previous study, is now presented to describe this minimal change.

Mathematically, let f be a function which maps STTs to the number of sessions; where w_i are weighting term. The objective becomes:

$$\min_{x_i} \{w_{s_1} \int_0^{x_i} f_{s_1}'''(x)^2 \, dx + w_{s_2} \int_{x_{i+1}}^{\infty} f_{s_2}''(x)^2 \, dx\} \quad (1)$$

In addition, the above model can be viewed as combining two different sets of workload information:

1. *Users that briefly visit the website:* These users visit the website for a very short period of time. Some of these users include users who do not plan to utilize the website. They are often directed to the website via a query entered into a search engine; however the site does not meet their requirements, but their decision cannot be made without them initially entering the site. Hence, on average, the sessions of users who briefly visit the site are extremely short and it is debatable whether these "visits"

should be considered as genuine sessions. Commonly these users only have a single interaction with the web-site; these visits can be viewed as having zero duration.

2. *Users that explicitly want to interact with the website:* These users actively interact with the website and often travel multiple web pages before finishing their session. Some of these users include users who are purchasing a product through an online store. These users will visit multiple web pages looking for product information before purchasing the item through an interactive shopping cart. Here the websites meet the users' requirements and provide information or services that actively engage the users. On average, these sessions are more extended than sessions of the first workload information.

It is far from obvious that this model should include data from both information sets; however, any partitioning is guaranteed to be less than perfect. The model can be trivially extended to accommodate this possibility by replacing the constant lower bound of the first integration term.

Finally a suitable basis for the weighting terms needs to be defined. Unfortunately, no obvious theoretical basis exists. S_2 should possess a significant length, and hence a weighting function as a function of the inverse of the length of this region seems appropriate. However, it is not clear that this concept yields any suitable formulation for S_1. For this study's estimations,

$$w_{s_1} = w_{s_2} = 1$$

Hence, the reformulated model is:

$$\min_{x_i} \{\int_{x_h}^{x_i} f_{s_1}'''(x)^2\,dx + \int_{x_{i+1}}^{x_j} f_{s_2}''(x)^2\,dx\} \quad (2)$$

where $h \geq 0; x_h < x_i < x_{j;}$ and $\int_{x_{j+1}}^{\infty} f_{s_3}''(x)^2\,dx = 0$

While the model is now complete, it clearly has limitations in terms of numerical stability given the estimation of higher-order differentials. For elongated data collection periods, this should not present a problem because the aggregated data will, in general, experience an averaging or smoothing effect. However, for short-term data, it is expected that the data will deviate from long-terms norms and can be viewed as more "noisy". Hence, there exists a strong possibility that such short-term data may require a smoothing approximation before the data is presented to the model. Based on the model discussed, a proposal is made to describe STT as the upper bound of the range denoted by the boundaries between S_1 and S_2, which is value at x_{i+1} in the presented model.

6.4 Description of the Websites under Investigation

This study will investigate server logs from two websites. The first website is a website for a company that specializes in online databases (Site A). This website is a commercial website that is very critical to Company A's operation. The

website utilizes the PHP (http://www.php.net) scripting language, MySQL[48] for the backend database and is hosted on an Apache HTTP Daemon[49]. In order to observe potential trends and patterns for this website, the log files chosen cover 27 months of operation from December 2004 to February 2007. This website represents a typical business website. That is, the site is a dynamic website with a mixed amount of static and dynamic pages – these are pages generated dynamically depending on the customers' inputs; its users are customers who are either looking to purchase a product or to register for a training course. The website contains several online databases. Users are charged for the time used to access these databases; these subscriptions are a main source of revenue for the company which is why the website is very critical to the organization. For the 27 months covered, Site A received approximately 3.6 million hits transferred 67 Gbytes of data.

The second website is www.ece.ualberta.ca, the website for the Department of Electrical and Computer Engineering at the University of Alberta. This site, although important to the organization, is non-commercial and not mission critical. This website is a dynamic website that utilizes the ColdFusion[50] scripting language, and the Apache HTTP Daemon. To investigate the data, the log files were chosen to cover 11 months of data. For this period, the ECE website received approximately 2.42 million hits, 203,896 "unique" visitors and transferred a total amount of 22.6 Gbytes of data. The data from this second website serves as a check to ensure that the trends observed with Site A are not unique to one particular website.

The log files under investigation are stored in the Common Log Format (CLF)[51] for ECE and the Combined Log Format[52] for Site A. To provide maximum flexibility with the analysis, a custom log parser was created in Ruby. All necessary information was extracted and imported into a DBMS. The approach used can be seen as a deep log analysis technique (Nicholas et al. 2000, 2006a, 2006b). The session length estimation requires at least two requests: one to mark start time of the session and one to mark the end time of the session. Hence, all users with only one request are removed from the log files. That is, all IPs that only have one record in the data are removed. A total of 10,938 users and 28,336 users are removed for Site A and ECE respectively.

Table 6.1 provides a summary of the properties of the logs used in previous studies and this study. Websites with an asterisk are commercial websites.

[48] http://www.mysql.com, last accessed February 7, 2010
[49] http://httpd.apache.org, last accessed February 7, 2010
[50] http://www.macromedia.com/software/coldfusion, last accessed February 7, 2010
[51] http://httpd.apache.org/docs/1.3/logs.html#common, last accessed February 7, 2010
[52] http://httpd.apache.org/docs/1.3/logs.html#combined, last accessed February 7, 2010

Table 6.1 Properties of log files used in previous studies

		Log duration	Requests	Bytes Transferred
Arlitt and Williamson (1997)	Waterloo	8 months	159 thousands	1.7 GB
	Calgary	1 year	727 thousands	7.6 GB
	Saskatchewan	7 months	2.4 millions	12.3 GB
	NASA	2 months	3.5 millions	62.5 GB
	ClarkNet*	2 weeks	3.3 millions	27.6 GB
	NCSA	1 week	2.5 millions	28.3 GB
Berendt et al. (2001)	University	12 days	175 thousands	n/a
Goševa-Popstojanova et al. (2006a)	NASA-Pvt1	20 weeks	23 thousands	0.5 GB
	NASA-Pvt2	20 weeks	92 thousands	0.2 GB
	NASA-Pvt3	20 weeks	489 thousands	2.2 GB
	NASA-Pub1	20 weeks	93 thousands	9 GB
	NASA-Pub2	20 weeks	732 thousands	6.7 GB
	NASA-Pub3	20 weeks	108 thousands	4.6 GB
	CSEE	6 weeks	5.8 millions	80.9 GB
	WVU	3 weeks	37.9 millions	97 GB
	ClarkNet*	2 weeks	3.3 millions	27.6 GB
	NASA-KSC	2 months	3.5 millions	62.5 GB
	Saskatchewan	7 months	2.4 millions	12.3 GB
Goševa-Popstojanova et al. (2006b)	WVU	1 week	15.8 millions	34.5 GB
	ClarkNet*	1 week	1.7 millions	13.8 GB
	CSEE	1 week	397 thousands	10.1 GB
	NASA-Pub2	1 week	39 thousands	0.3 GB
He and Goker (2000)	Excite*	50 minutes	51 thousands	n/a
	Reuters*	9 days	9.5 thousands	n/a
Huntington et al. (2008)	OhioLINK	12 months	n/a	n/a
Tian et al. (2004)	SMU/SEAS	26 days	763 thousands	7.8 GB
	KDE	31 days	14 millions	110 GB
This study	Site A*	27 months	1.9 millions	33.5 GB
	ECE	11 months	2.4 millions	22.6 GB

This table shows that the longest period that previous studies have collected data is over a 12 month period, compared to 27 months in this study. Furthermore, many of the previous studies are not performed on a commercial website. For studies that use logs from commercial websites, the periods covered are extremely short (50 minutes to 2 weeks). This study investigates the log file from a commercial website for a much longer period (27 months). Hence, it is believed that this study presents the first long-term analysis of a commercial website. This long data period provides several benefits over short data periods.

- A short data collection period cannot truly capture users' behaviors because their behavior is by definition variable and only a single snapshot

of their behavior is likely to be captured with the short data collection period. For example, a new user to a Wiki may simply read articles, once familiar with the website the user may choose to participate in other activities such as posting comments, providing feedback or even editing articles.

- An organization's behavior also affects its website traffic patterns. Advertising campaigns, various public announcements will often increase the amount of traffic. For example, GoDaddy.com's website experienced a 1500 percent increase in traffic following its Super Bowl ad campaign. Other websites advertised during Super Bowl Sunday also had their traffic increased. Short term collection either overstates these actions if it is performed near a major activity or understates them if performed far from the activity.
- Well known trends and periodic patterns such as the "weekend effect" will distort short term collection resulting in skewed data. In fact, Arlitt and Jin (2000) have demonstrated that websites have very different workload intensities on weekdays versus weekends. Hence, if the data period is short, the analysis will be skewed by such effects.
- Major web events will also affect the data sets gathered within a short time frame. For example, popular YouTube videos are known to result in millions of hits to YouTube's website within a short period of time before the site's traffic returns to normal. A website being mentioned on another popular website such as Slashdot will also cause the website's traffic to increase. This is commonly known as the Slashdot effect.
- Short collection periods can experience distortion due to either higher than normal or lower than normal activities from robots. For example, many ticket scalpers use robots to automatically purchase event tickets from Ticketmaster when they first go on sale. This is especially true for popular events where tickets can be sold within minutes of being available online. Short collection periods would result in skewed data from the activities of these robots.
- Users have very low brand loyalty. If quality of service (such as response period) is poor, users leave quicker than normal (the inverse will be at some-level true) – this impacts session statistics and again short-collection periods can get skewed because of the quality of service differing from the long-term norm. For example, a user many visit a website during maintenance which may cause the website to response much slower than usual. The quality of service during this maintenance period cannot be considered as the normal QoS for the website.

6.5 STT Results and Discussions

In order to apply the model discussed, the effects that different threshold values have on the total number of sessions are calculated for the two websites under investigation. The explored STT values are from 1 to 120 minutes in 1 minute intervals. To search for repeating effects, four different resolutions are investigated: days, weeks and months.

6.5.1 Removing Automated Requests

While applying the model, it was discovered that robots and other automated systems used to request resources need to be removed from the server logs in order for the model to be used effectively. That is, systems that automatically request a resource from the website after a set period of time will cause the model's description of the regions to be incorrect. For example, Figure 6.1 shows the number of sessions versus STT before the removal of several site monitoring systems from the log files for Site A.

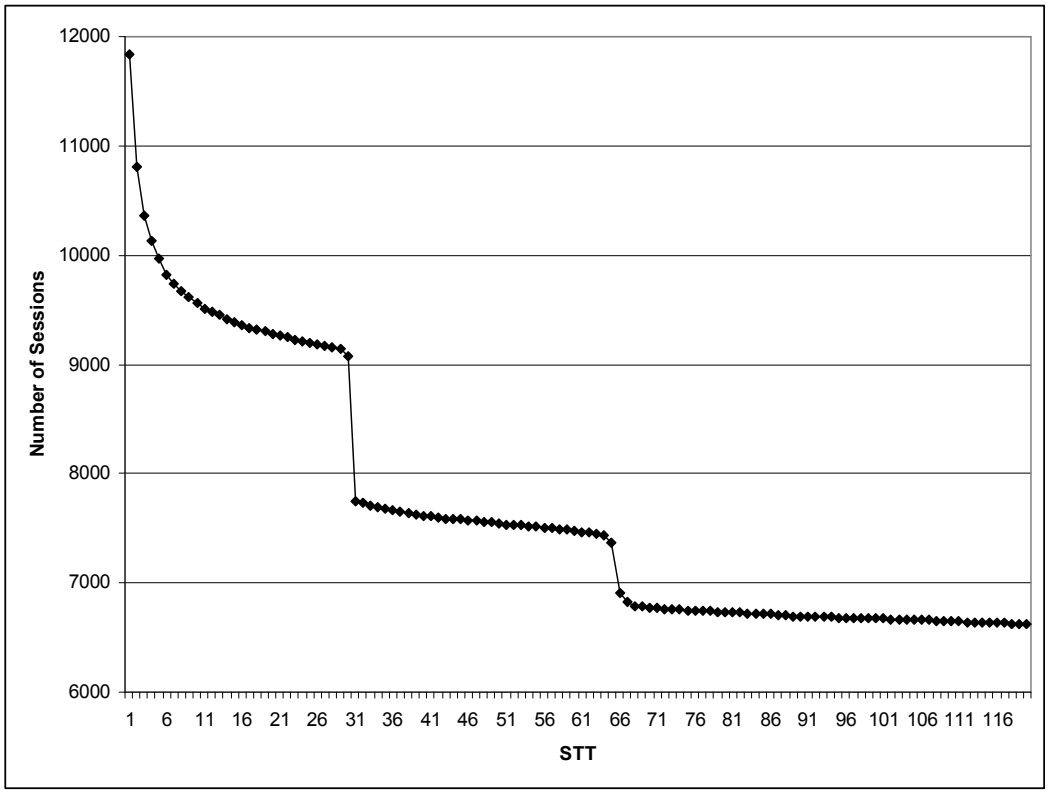


Figure 6.1 Number of sessions versus STT before removal of monitoring systems

This figure shows many distinct regions. Close examination of the server log reveals that two monitoring services are used to monitor the website's status. The first service requests a resource from Site A every 30 minutes while the second service requests a resource from Site A every 66 minutes. Removal of these records from the server logs was simple because the resources these services request are unique and are not publicly available. ECE's log files were also parsed to remove robots that automatically request the "robots.txt" resource every 60 minutes.

Although, it is infeasible to remove all automated requests from the server logs, web administrators need to remove all identifiable requests. Several techniques to identify them can be used by web administrators to remove automated requests.

Most well known robots have a signature line that is included with every request as part of the USER AGENT field of the log file. For example, "Googlebot-Image/1.0" can be used to identify a robot from Google that is indexing the website's images. For web monitoring services, web administrators can simply dedicate a special resource that only these services can access. This resource can then be easily identified within the log files. Armed with adequate information, web administrators can eliminate most automated requests from the server logs which will enable the STT to be estimated more accurately. For the two websites under examination, 77,530 automated "users" are removed from ECE and 34,625 automated "users" are removed from Site A.

6.5.2 Day Resolution Investigation

One hundred weekdays and fifty days on the weekends for each website were randomly chosen for this investigation. Two sample day graphs for the websites can be seen in Figures 6.2-6.3.

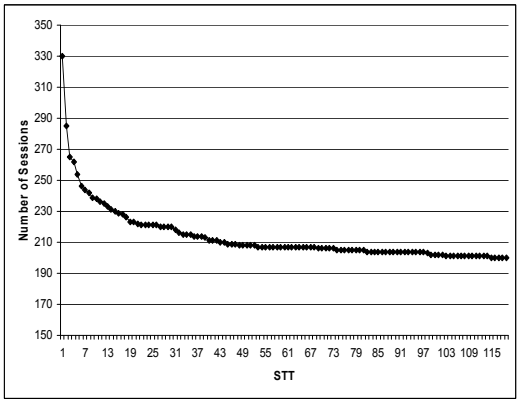


Figure 6.2 A random Site A day

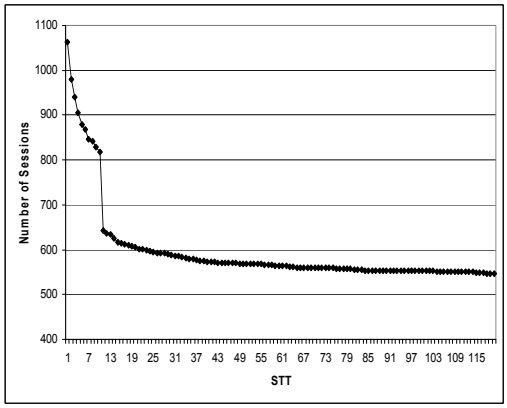


Figure 6.3 A random ECE day

Informal observations show distinct surface regions in Figure 6.2 and 6.3 which suggests that the model is applicable at this resolution. Hence, the model was applied and the x_{i+1} (STT) results for each website were obtained. Unfortunately, nothing is known about the distributional characteristics of this value, and hence both parametric and non-parametric measures are utilized to explore this concept. The results, presented in Table 6.2 and Figures 6.4 to 6.7, show that the two websites have very different workload intensities and behaviors, which suggests that they are unlikely to share the same STT value.

Table 6.2 STT for day resolution

	Site A - Weekdays	Site A - Weekends	ECE - Weekdays	ECE - Weekends
Mean	5.24	3.68	9.72	8.96
Mean st.err	0.213	0.160	0.392	0.517
Median	5	4	9	8
Variance	4.528	1.283	15.396	13.386
St.Dev	2.128	1.133	3.924	3.659
Minimum	2	2	2	2
Maximum	12	6	25	16
Range	10	4	23	4
Skewness	0.886	0.322	0.776	0.111
Skewness st.err	0.241	0.337	0.241	0.337
Kurtosis	0.379	-0.377	1.828	-0.580
Kurtosis st.err	0.478	0.662	0.478	0.662

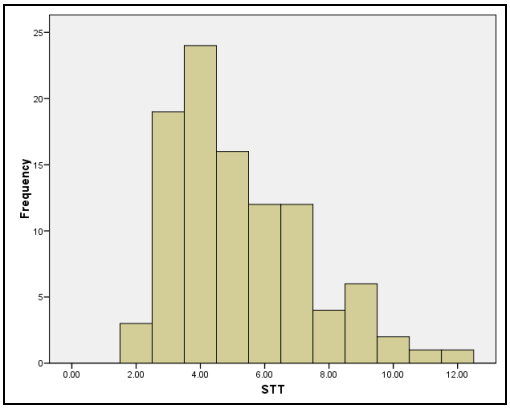


Figure 6.4 STT Histogram for Site A at Weekdays Resolution

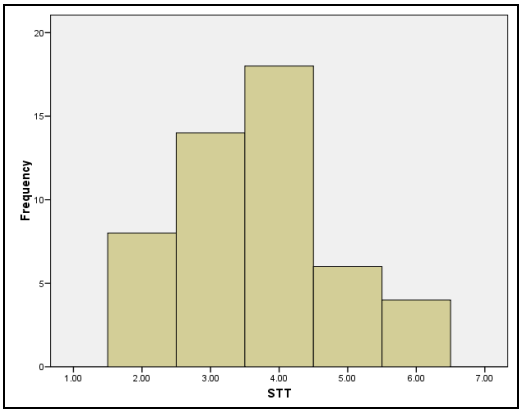


Figure 6.5 STT Histogram for Site A at Weekends Resolution

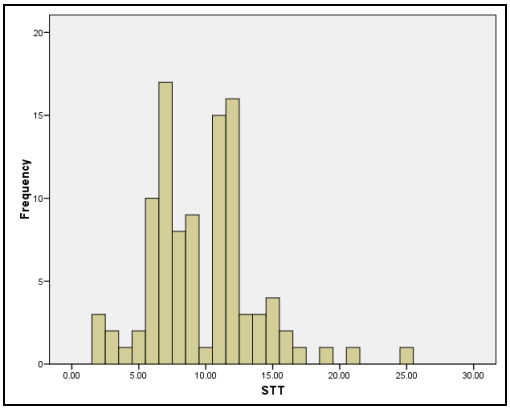


Figure 6.6 STT Histogram for ECE at Weekdays Resolution

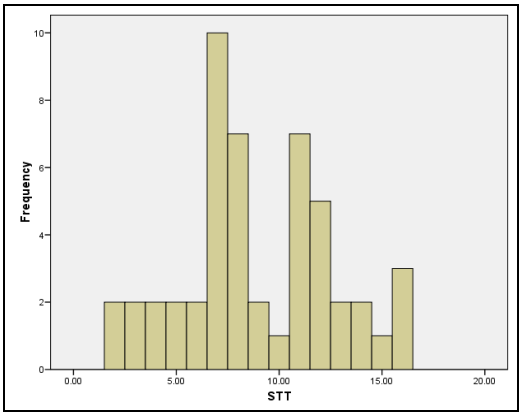


Figure 6.7 STT Histogram for ECE – Weekends Resolution

These results show that the values at x_{i+1} are not stable and vary depending on the day under observation. This is expected because web workloads (similar to other workloads) have day and weekly periodicity; hence different days in the week usually have different workload behaviors and intensities. In order to determine differences between the weekdays versus weekends data sets, an F-test was performed using 50 random samples from the weekday and weekend data sets. Table 6.3 displays the results.

Table 6.3 F-Test

	F	*p*-value	d.f.
Site A	3.8664	< 0.001	49
ECE	0.9599	0.44	49

The results show that the null hypothesis (the weekdays and weekends workloads are the same) can be rejected for Site A. However, for the ECE data sets, the results are much less clear.

The mean x_{i+1} value for Site A is less than ECE; furthermore, Site A, with a standard deviation of 2.13 for weekdays and 1.13 for weekends, has a tighter spread of values compared to ECE. The STT is smaller on weekends for both websites which further confirms web traffic generalization as discussed by Arlitt and Jin (2000) and Pitkow (1999). The differences in results may be attributed by the different user profiles the two websites experienced. A hypothesis can be proposed based on the fact that users are charged for usage time for Site A, hence, the STTs are generally shorter. Clearly, this hypothesis needs more exploration before it can be confirmed.

The results seem to empirically demonstrate that the threshold does exist for this "fine resolution" data; while some variation exists the results by no means look random or unsystematic. Using SPSS, several Q-Q (Quantile-Quantile) plots for the following distributions were examined: Chi-square, Exponential, Gamma, Half-normal, Laplace, Lognormal, Normal, Pareto, Student t, Weibull, and Uniform. The estimated STT data shows a "reasonable" fit to several of the distributions; Figures 6.8 and 6.9 show the Q-Q plots for two "possible" distributions, Normal and Gamma respectively.

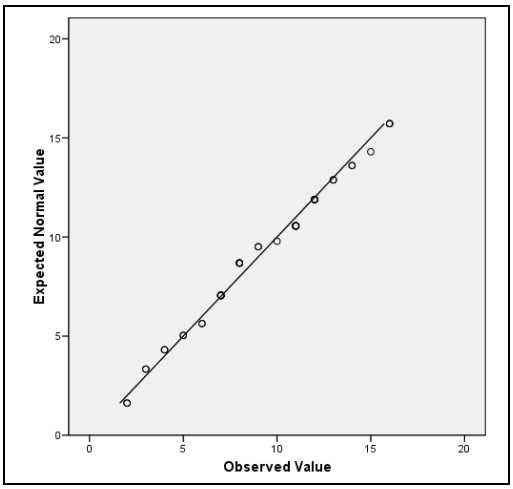


Figure 6.8 Normal Distribution Q-Q plot for ECE

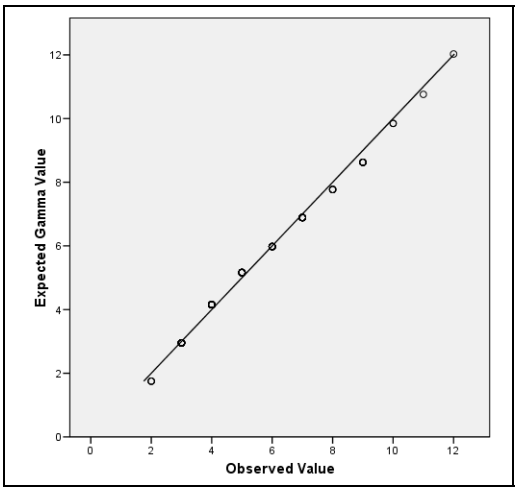


Figure 6.9 Gamma Distribution Q-Q plot for Site A

This possibility is explored more formally, but only for a normal distribution. The Shapiro-Wilk test (Shapiro and Wilk 1972) was applied to the data sets. This test was selected as it tends to be more powerful than other common normality tests, such as Anderson-Darling and Kolmogorov-Smirnov (Stevens and D'Agostino 1986), and does not require that the mean or variance of the hypothesized normal distribution to be specified in advance. Table 6.4 displays the results using the Shapiro-Wilk test function in SPSS. The results show that three of the four *p*-values for both websites are less than 0.05; hence, these three data sets are (probably) not normally distributed.

Table 6.4 Shapiro-Wilk test

	Coefficient	*p*-value	d.f.
Site A - Weekdays	0.92	< 0.001	99
Site A - Weekends	0.91	< 0.001	49
ECE - Weekdays	0.95	< 0.001	99
ECE - Weekends	0.97	0.149	49

6.5.3 Week Resolution Investigation

In order to examine the week resolution, 50 random weeks were chosen for Site A and 25 random weeks were chosen for ECE. Figures 6.10 and 6.11 show the number of sessions versus STT for two of the weeks.

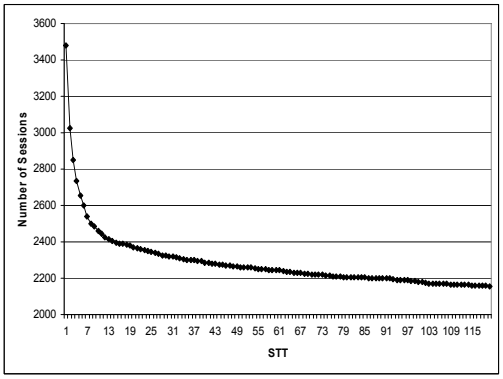


Figure 6.10 A random week for Site A

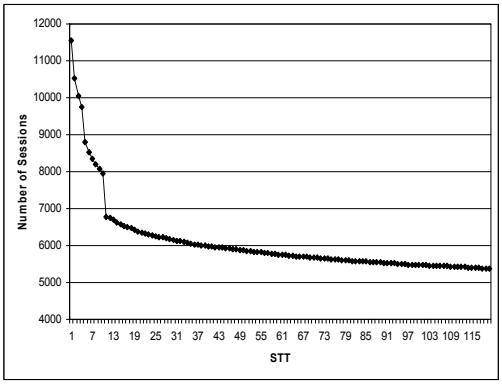


Figure 6.11 A random week for ECE

As was expected, on average, the week resolution curves are smoother than the day resolution curves. However, Figures 6.10 and 6.11 still clearly display distinct surface regions which, again, suggest the applicability of the model. The x_{i+1} (STT) results from applying the model to these random weeks are shown in Table 6.5 and Figures 6.12 – 6.13.

Table 6.5 STT for week resolution

	Site A	ECE
Mean	4.571	8.002
Mean st.err	0.157	0.361
Median	7	5
Variance	1.208	3.250
St.Dev	1.099	1.803
Minimum	3	6
Maximum	8	12
Range	14	6
Skewness	0.111	1.067
Skewness st.err	0.340	0.464
Kurtosis	1.107	-0.162
Kurtosis st.err	0.668	0.902

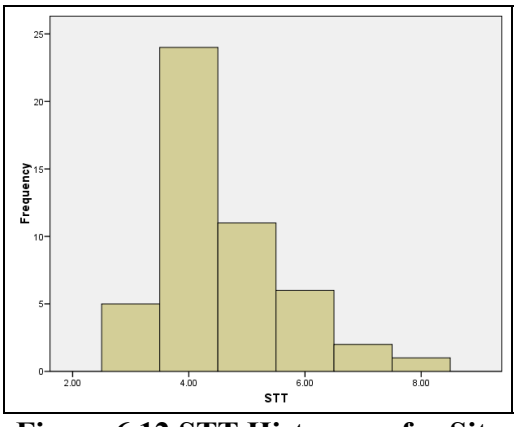


Figure 6.12 STT Histogram for Site A at the Week Resolution

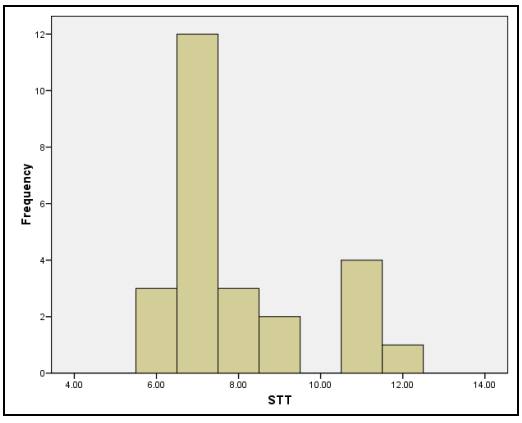


Figure 6.13 STT Histogram for ECE at the Week Resolution

These results show that the STT for both websites have settled between the STT for weekdays and weekends. This is expected because of the averaging effect. However, Site A and ECE still have different STT values, which mean that the STT should not be a single constant for all websites at this resolution. Q-Q plots for several distributions at this resolution have broadly the same results as the day resolution.

Table 6.6 Shapiro-Wilk test for the week resolution

	Coefficient	*p*-value	d.f.
Site A	0.85	< 0.001	49
ECE	0.80	< 0.001	24

Table 6.6 shows results from the Shapiro-Wilk test which again implies that both data sets possess data which is probably non-normal.

6.5.4 Month Resolution Investigation

Figures 6.14 and 6.15 show two sample graphs for ECE and Site A at one month resolution. In total, all 27 months were investigated for Site A, and all 11 months were investigated for ECE.

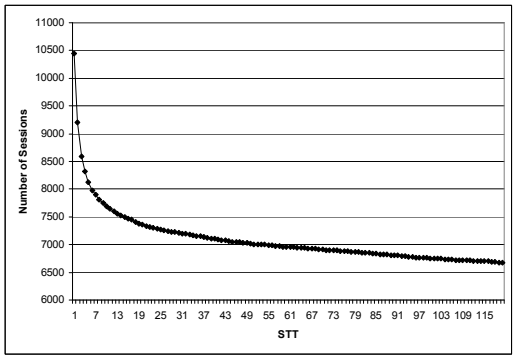


Figure 6.14 A random month for Site A

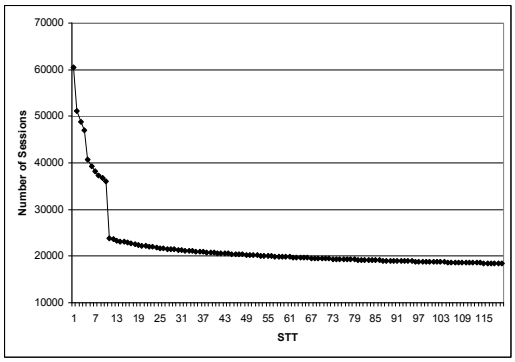


Figure 6.15 A random month for ECE

Initial observations show that Figure 6.14 displays a seemingly smooth curve due to the averaging effect. However, Figure 6.15 displays visible distinct surfaces as described in the model. Determining *t* by casually observing the figures is now difficult and error-prone. Using the provided model, the x_{i+1} (STT) values can be calculated for all the graphs. Table 6.7 and Figures 6.16-6.17 display the results from the calculations. Q-Q plots for various distributions at this resolution again show broadly the same results as obtained in the week and day resolutions.

Table 6.7 STT for month resolution

	Site A	ECE
Mean	5.001	8.455
Mean st.err	0.207	0.638
Median	5	7
Variance	1.154	4.473
St.Dev	1.074	2.115
Minimum	4	6
Maximum	7	12
Range	3	6
Skewness	0.402	0.659
Skewness st.err	0.448	0.661
Kurtosis	-1.414	-1.359
Kurtosis st.err	0.872	1.279

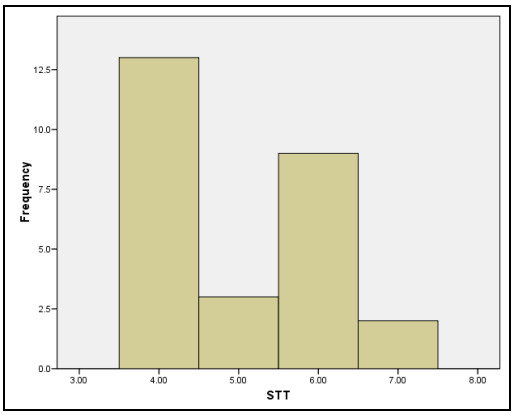


Figure 6.16 STT Histogram for Site A at the Month Resolution

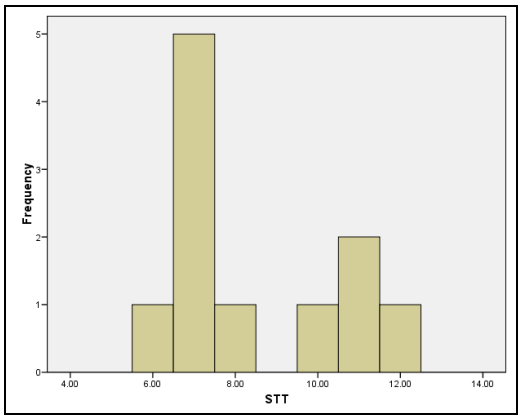


Figure 6.17 STT Histogram for ECE at the Month Resolution

Table 6.8 Shapiro-Wilk Test for the month resolution

	Coefficient	*p*-value	d.f.
Site A	0.79	< 0.001	26
ECE	0.83	0.023	10

The results show that both the variation and the range are reduced with the increase in the collection period; and that the possibility exists that any estimation of *t* at this level of resolution might be considered as a "long-term" norm. However, this can only safely be performed if the administrator knows that there have been no major modifications to the website, which may affect its users' behaviors or the usability of the website. That is, if the operational profile of the website remains stationary. For example, an online store may add a "Users' Review" section which causes users to spend more time at the store to read the reviews. If an administrator uses a one month period before this feature is added, the *t* value calculated from x_{i+1} will be different than the one month period after this feature is added. Table 6.8 shows the results from the Shapiro-Wilk test; and

at this resolution, data from both web sites needs to be considered as again (probably) non-normal. An investigation of the behaviour of STT for the total period of investigation for both web sites repeats the previous patterns; the details are omitted for the sake of brevity.

This page is intentionally left blank.

Chapter 7 – Investigating the Distributional Property of the Session Workload

Many researchers have investigated the session workload. However, the investigations into the distributional properties of the session workload lack rigorous analysis. In fact, Goševa-Popstojanova et al. (2006a, 2006b) is the only known study to provide a detailed analysis of the measure's characteristics. However, this study only considers "are session lengths sampled from a heavy-tailed distribution" without convincing evidence that this characterization is definitive. The implications of whether the session length is heavy-tailed can have a significant impact on the formulation of many website models. For example, Tian et al. (2004) proposed a reliability model for websites based on a short-tailed distribution which would be invalid if the session length is heavy-tailed. Furthermore, let's consider constructing a simple reliability model of a website. If we assume that the probability of any software failure per input or hit is constant, p, we have a simple binomial process. The number of failures f_n after n inputs is given by the binomial distribution:

$$P(f_n = k) = \binom{n}{k} p^k (1-p)^{n-k} \quad (1)$$

Therefore, the probability of the system failing after n hits occurs whenever $f_n > 0$. Hence,

$$\begin{aligned} P(f_n > 0) &= 1 - P(f_n = 0) \\ &= 1 - (1-p)^n \end{aligned} \quad (2)$$

The system administrator might want to think about the defect rate of the system as a function of time rather than as a function of the number of inputs or hits.

$$1 - (1-p)^n = 1 - (1-p)^{At} \quad (3)$$

where A is the average inputs per time unit (t). Further, considering the data presented in the previous sections, p is obviously small and n is obviously large allowing the binomial process to be approximated by an exponential distribution.

$$= 1 - e^{-Atp} \quad (4)$$

If the distributional property is heavy-tailed, this model would be invalid because the average inputs per time unit A is infinite. These types of models are neither new nor unique to reliability. Many dynamic characteristics of websites may be approximated by such models. However, if the workloads are heavy-tailed, many of these models will be invalid because either the mean or the variance is infinite. That is, they require an estimation of one of the moments of the workload variable; yet, the moments are infinite in heavy-tailed distributions.

The session workload unit has also been used to mine web usage for web personalization (Eirinaki and Vazirgiannis 2003, Mobasher et al. 2000). This personalization process allows websites to customize themselves to match the users' usage patterns. For example, Amazon.com uses web mining data from user sessions to recommend books to their customers. Jasen and Spink (2003)) examined user sessions to determine how web search engines are utilized and which search results are being viewed by the users. Cherkasova and Phaal (2002) proposed a session-based load management for commercial websites to improve quality of service; they utilized a simulation to model the session workloads in their study. All approaches mentioned are dependent on the session workload model. Hence, the acceptance of the conjecture that workloads are sampled from heavy-tailed distribution has serious ramifications for future research and analysis of the "behavior" of websites. Therefore, this chapter re-evaluates the results presented by Goševa-Popstojanova et al. (2006a, 2006b) which conclude that session length data is sampled from a heavy-tailed distribution. The conclusion was based on results from the analysis of the log-log complementary distribution plots (LLCD) and the Hill estimator. However, a more rigorous empirical investigation into session length and its potential distributional properties can be performed.

This chapter extends Goševa-Popstojanova et al. (2006a, 2006b) by applying the evaluation to two new websites. One of which is a mission-critical commercial website. The logs investigated for this commercial website cover a 27 month period, an extensive time period. Other investigations are "focused" on high throughput web sites for a short period. However, examining a website over a long calendar period is essential as many "external actions" which impact the characteristics of the site happen infrequently as hence a true sense of the historical norm of a website's characteristics is only available over an extended period. A more detailed discussion of the two websites can be seen in Chapter 6, Section 6.3.

Additional tests, such as the Heavy-tailed Autocorrelation Function (ACF) method, "wobble analysis" and Q-Q (Quantile-Quantile) plots, are performed to determine if session length can really be modeled by a heavy-tailed distribution. The results from this chapter show that, for the samples used in this study, a method to determine whether the session workload can be modeled by a heavy-tailed distribution does not exist.

The remaining sections of this chapter are organized as follows: Section 7.1 re-evaluates the heavy-tailed property of session lengths. It investigates the validity of using log-log complementary distribution (LLCD) plots and the Pareto distribution to model the session length as presented by Goševa-Popstojanova et al. (2006a, 2006b). Section 7.2 discusses the results from this study versus the previous study.

7.1 Investigation of the Distributional Characteristics of Session Length

Goševa-Popstojanova et al. (2006a, 2006b) put forward the conjuncture that session length data is sampled from a heavy-tailed distribution. In this section this conjecture is empirically examined.

7.1.1 Discussion of the STT

This study uses a Session Timeout Threshold to determine the sessions. A session is defined as a sequence of actions taken by a user within a period of time. Sessions offer much finer grained information than the standard *number of users* metric. Goševa-Popstojanova et al. (2006a, 2006b) assign STT to 30 minutes, because it is a common value used by other researchers (Berendt et al. 2001, Mahoui and Cunningham 2000, Mat-Hassan and Levene 2005, Spiliopoulou et al. 2003). This 30 minute figure is a value rounded up based on a mean value of 25.5 minutes determined by Catledge and Pitkow (1995). Catledge and Pitkow (1995) estimate STT to be 25.5 by claiming that the most statistically significant events occurred within 1.5 standard deviations (25.5 minutes) from the mean between each user interface event which was 9.3 minutes. However, no definition of these "significant events" was given; and why 1.5 standard deviations is selected is never discussed. Hence, this study uses a model proposed by Huynh and Miller (2009) to determine the STT. By applying the model, the STT is found to be 5 minutes for Site A and 11 minutes for ECE. As a cross-check, the results presented in this study were replicated using STT = 30 minutes for both sites; and while the numerical values clearly changed the basic interpretation of the results remained constant.

7.1.2 Estimating the Tail Index α with LLCD Plot

Under the assumption that the data comes from a Pareto distribution, Goševa-Popstojanova et al. (2006a, 2006b) estimate the tail index of the distribution using a log-log complementary distribution (LLCD) plot. This approach has also been used in many studies which concentrate on other workload metrics for web servers (Arlitt and Jin 2000, Arlitt and Williamson 1997, Crovella and Bestavros 1997). LLCD plots produce an estimate of the tail index using the property

$$\frac{\partial \log(P[X > x])}{\partial \log x} = -\alpha \quad (7)$$

However, the approach does not utilize the entire distribution. The estimation of the index is only over the range $[x_i, x_{i+j}]$; and the approach simply fits an ordinary least-squares linear regression model to estimate α from the small set of values ($[x_i, x_{i+j}]$) which are assumed to represent the majority of the tail.

Downey (2001a, 2005) has shown that the LLCD plot is an ineffective mechanism at discovering long-tailed distributions. Basically, the technique cannot adequately distinguish between long-tailed distributions, such as the Pareto distribution, and "similar looking" short-tailed distributions such as lognormal

distributions. Figueiredo et al. (2005) further support this viewpoint and provide an extensive analysis demonstrating the inadequacy of this approach; they demonstrate that the discovery of the appearance of a linear region in a LLCD plot is by itself insufficient evidence to conclude that long-range dependence exists within a data set. Finally, Goldstein et al. (2004) empirically demonstrate that the LLCD plot and associated techniques are ineffective approaches to fitting power-law distributions to experimental data and conclude that the approach should be avoided.

7.1.2.1 Discussions of the LLCD Plot Results

This study uses three definitions of the tail as presented by Hernandez-Campos et al. (2004). The *extreme tail* is the part of the tail that is beyond the last data point (x_n), hence no information is available for this part. The *far tail* is the part of the tail where some data is present, but the distributional properties cannot be understood because of the minimal information available (around x_{i+j}). The *moderate tail* is the part of the tail that contains "rich" (by comparison) distributional information ($[x_i,x_{i+j}]$). Clearly, the definitions are heuristics because the boundary between the *moderate tail* and the *far tail* cannot be defined accurately. However, the definitions are required for discussions of the results in this section.

Goševa-Popstojanova et al. (2006a, 2006b) have estimated α using LLCD plots. Figures 7.1, 7.4, 7.7 and 7.10 display the LLCD plots for ECE and Site A with each having two different STT values. This study utilized Huynh and Miller (2009) dynamic STT estimation model and the commonly used 30 minute constant STT value approach used by Goševa-Popstojanova et al. (2006a, 2006b) to investigate if the value of STT was a covariant of the distributional characteristics of the session length. Hence, LLCD plots were created for both the dynamic model's STT values and the constant STT value. These figures show that for values below −1 on the vertical axis the distribution is generally linear until the far tail is reached. Although, linear least squares fitting can be applied to estimate α, this study uses a numerical differential equation to estimate α at all possible data points. Figures 7.2, 7.5, 7.8 and 7.11 show results of this estimation. These figures show that α does not stabilize in the moderate tail in any of the LLCD plots. The variations are consistently too large to be explained by numerical differential estimation error. To further confirm this observation, the box plot for α for all LLCD plots are shown in Figures 7.3, 7.6, 7.9, and 7.12, and the descriptive statistics for α are shown in Table 7.1. Box plots are used in this study for their ability to visually display different types of populations without any dependency on the statistical distribution of the data. These figures show that the range for the non-outliers varies considerably; furthermore, the outliers are numerous. Figures 7.2 and 7.8 perhaps provide the clearest evidence of α failing to stabilize within the tail of the distribution. These figures can be approximated as:

1. estimates for α are relatively "well-behaved" in the *pre-tail*;
2. estimates for α vary wildly in the *moderate tail*; and

3. estimates for α seem to be almost random values in the *far tail*.

Because the type of distribution for the data sets is unknown, Table 7.1 displays the statistics for both parametric and non-parametric distributions. This table can be seen as an exploratory tool to aid the data examination process. The table and box plots further confirm that α is not stable enough for the least-squares linear regression model

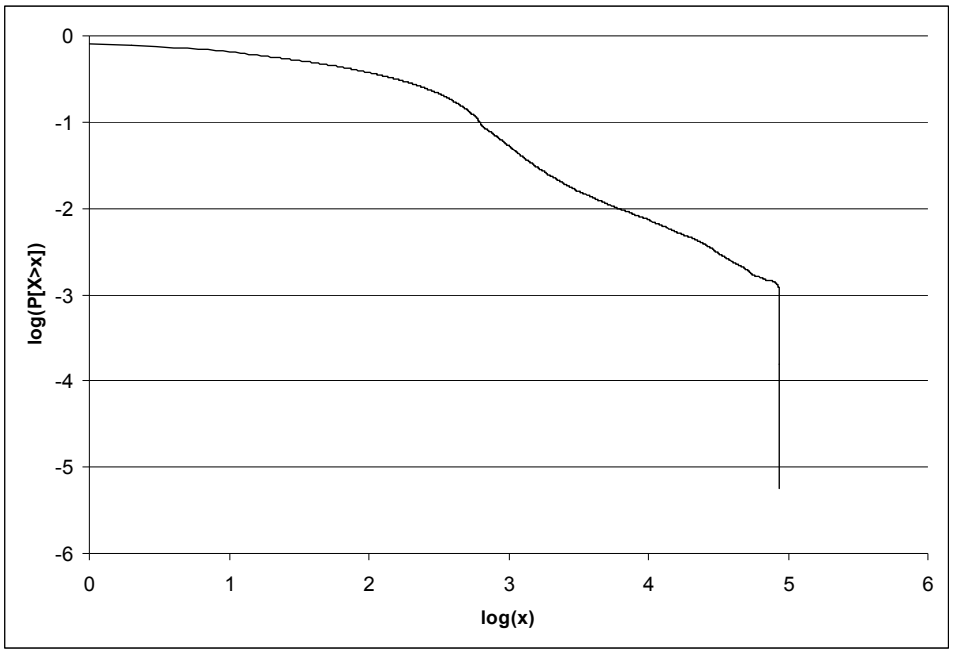


Figure 7.1 LLCD Plot for ECE with 11mins STT

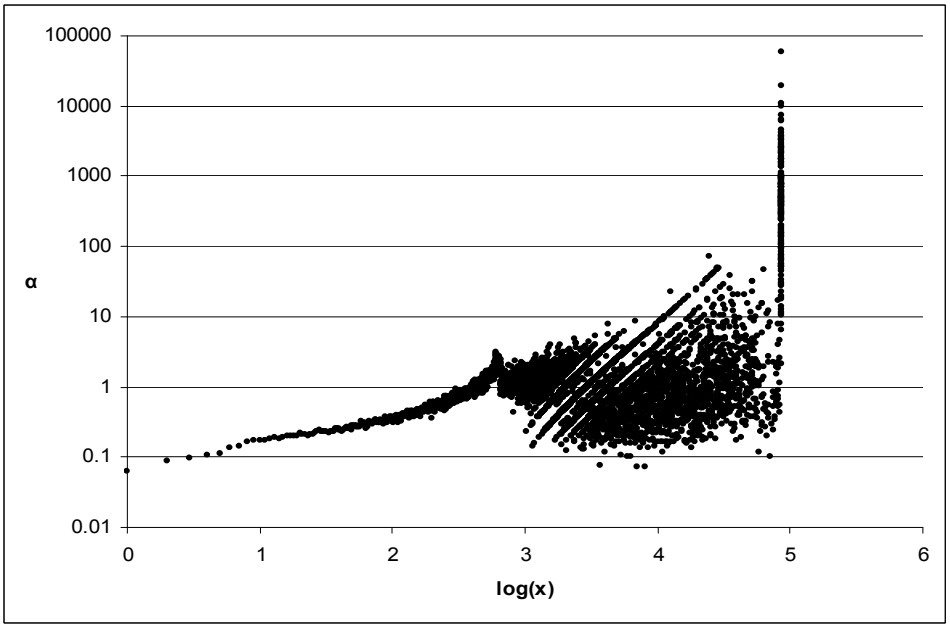


Figure 7.2 Numerical Differential Estimation of α for ECE with 11mins STT

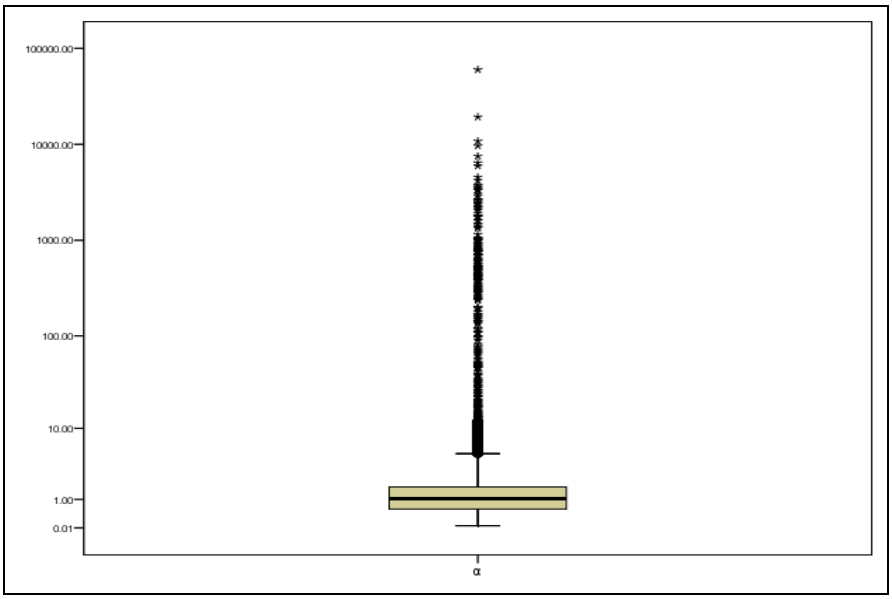


Figure 7.3 Box plot of α for ECE with 11mins STT Showing Numerous Outliers

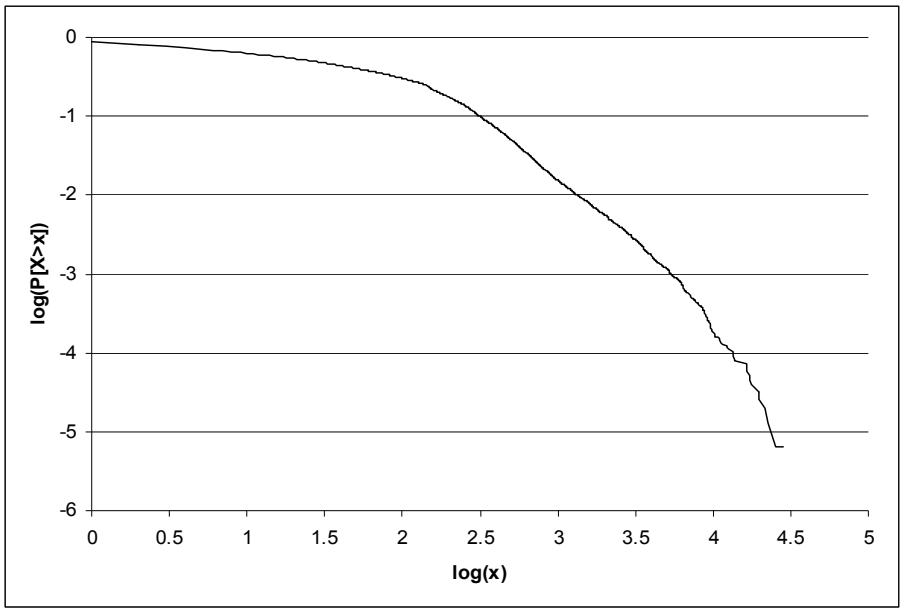


Figure 7.4 LLCD Plot for Site A with 5mins STT

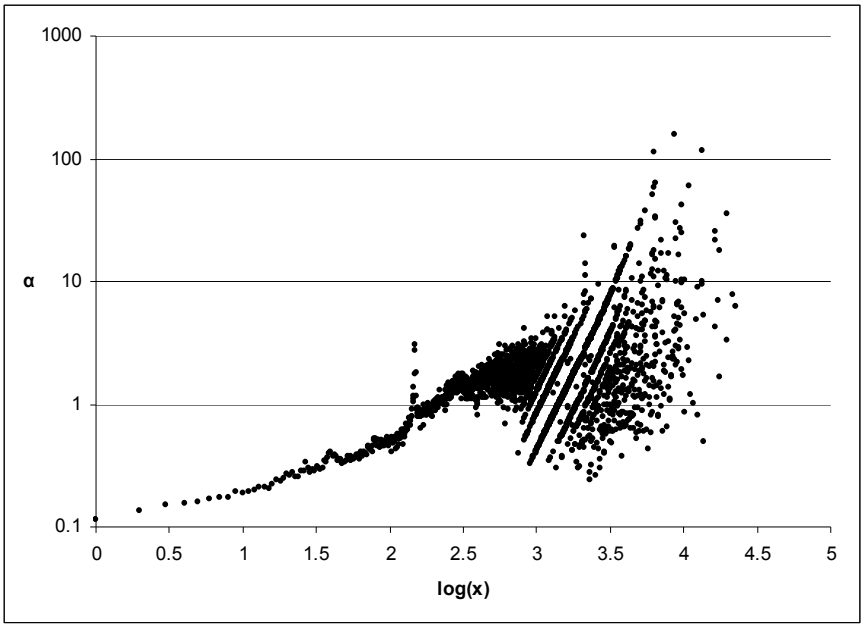


Figure 7.5 Numerical Differential Estimation of α for Site A with 5mins STT

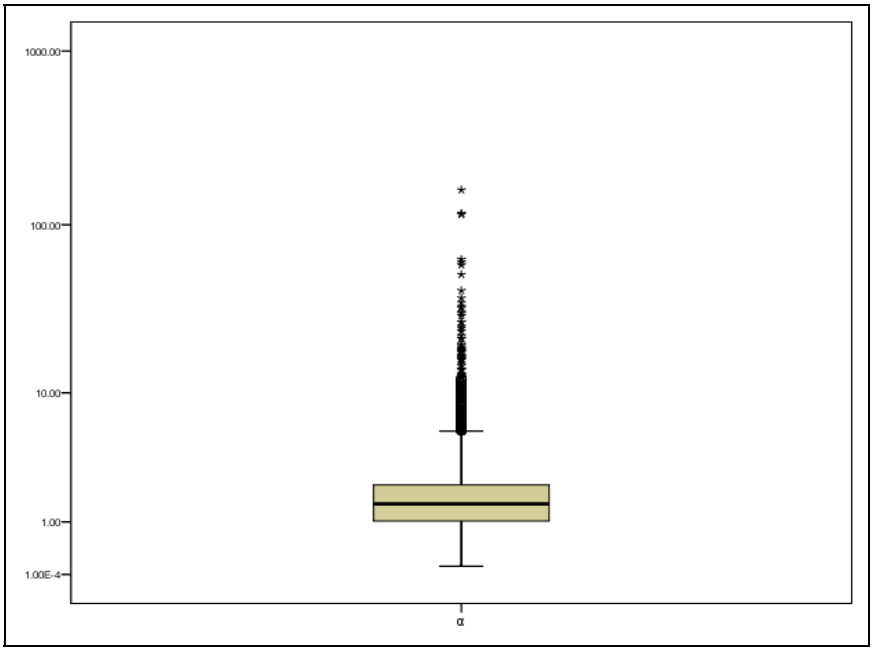


Figure 7.6 Box plot of α for Site A with 5mins STT Showing Numerous Outliers

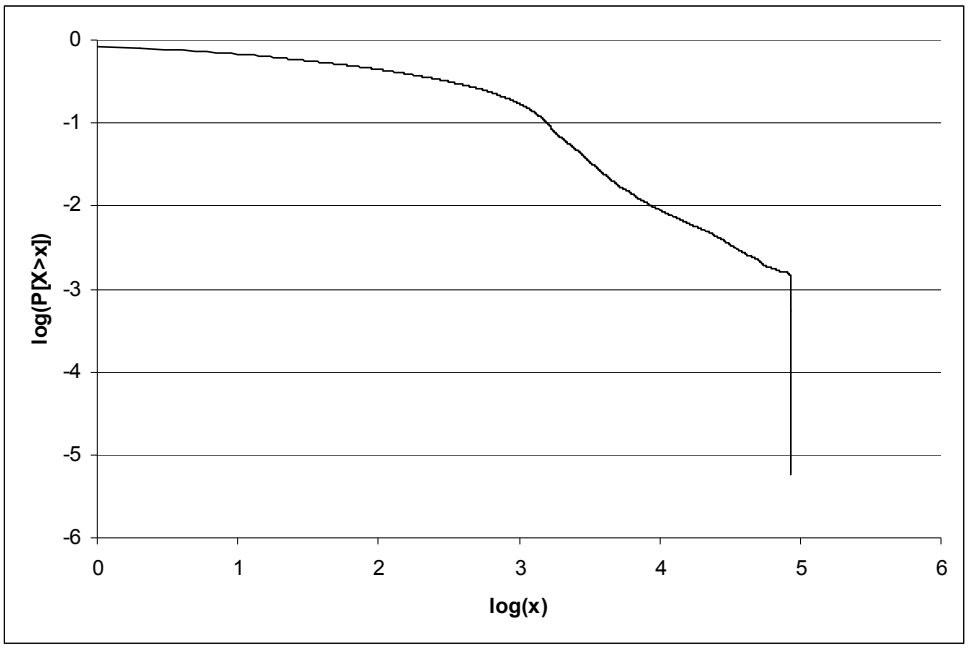


Figure 7.7 LLCD Plot for ECE with 30mins STT

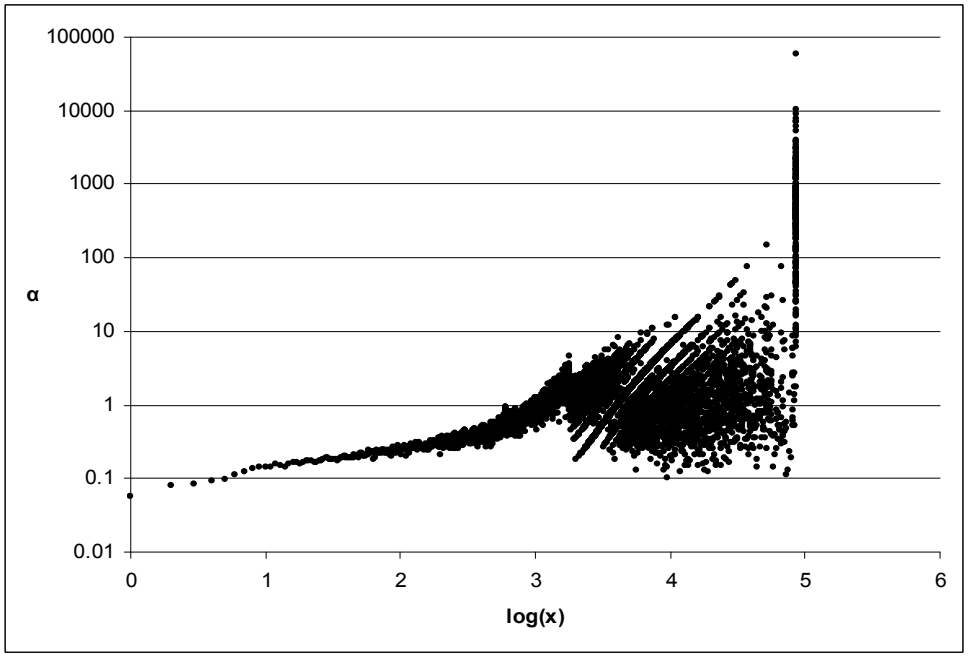


Figure 7.8 Numerical Differential Estimation of α for ECE with 30mins STT

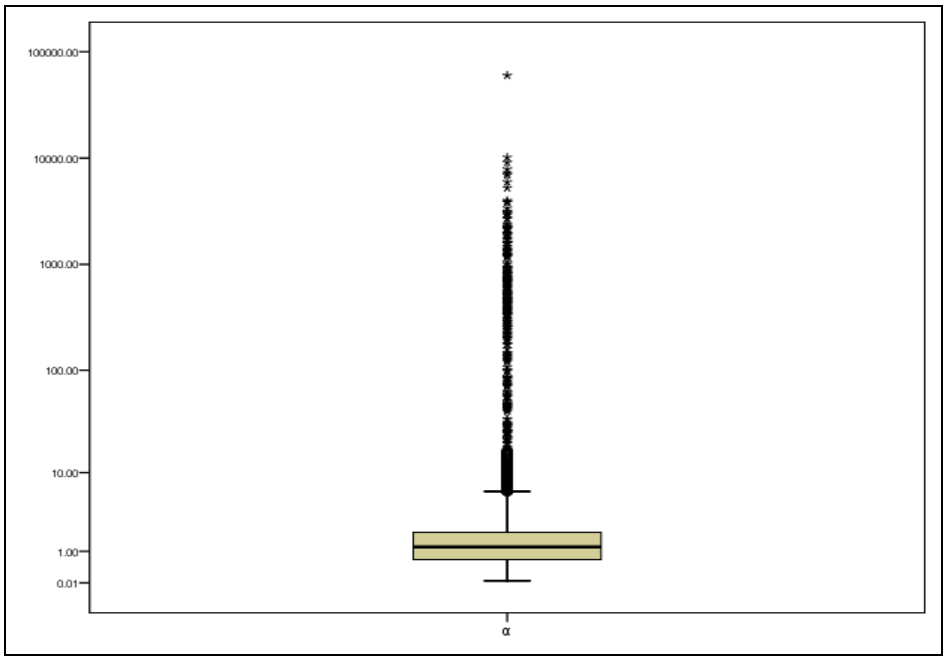


Figure 7.9 Box plot of α for ECE with 30mins STT Showing Numerous Outliers

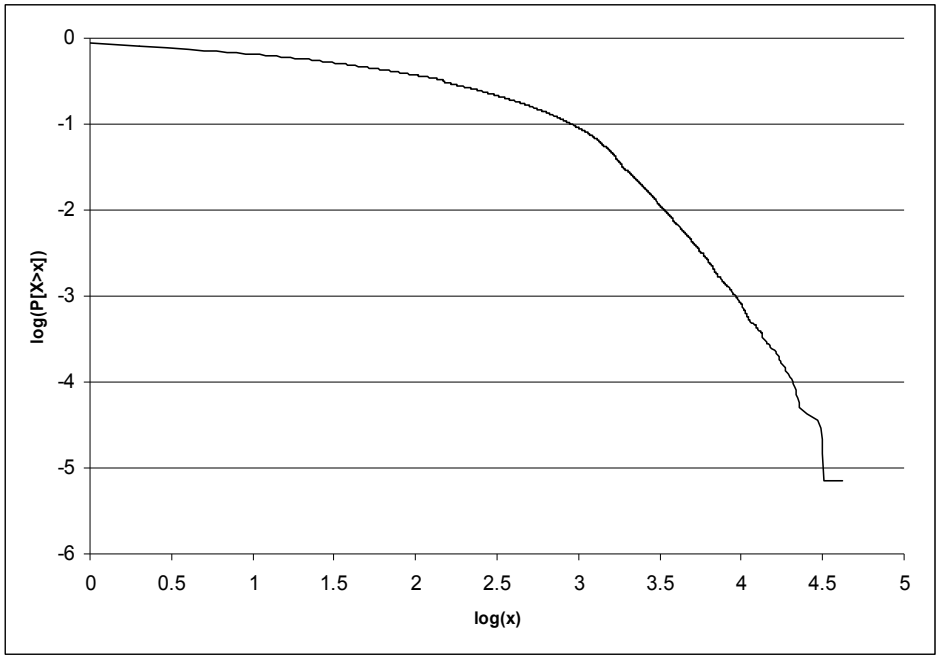


Figure 7.10 LLCD Plot for Site A with 30mins STT

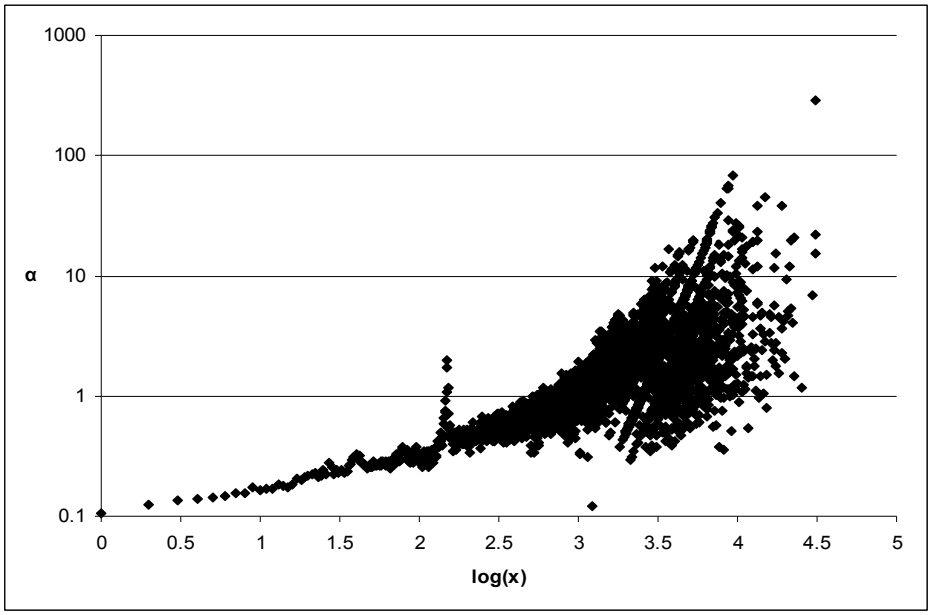


Figure 7.11 Numerical Differential Estimation of α for Site A with 30mins STT

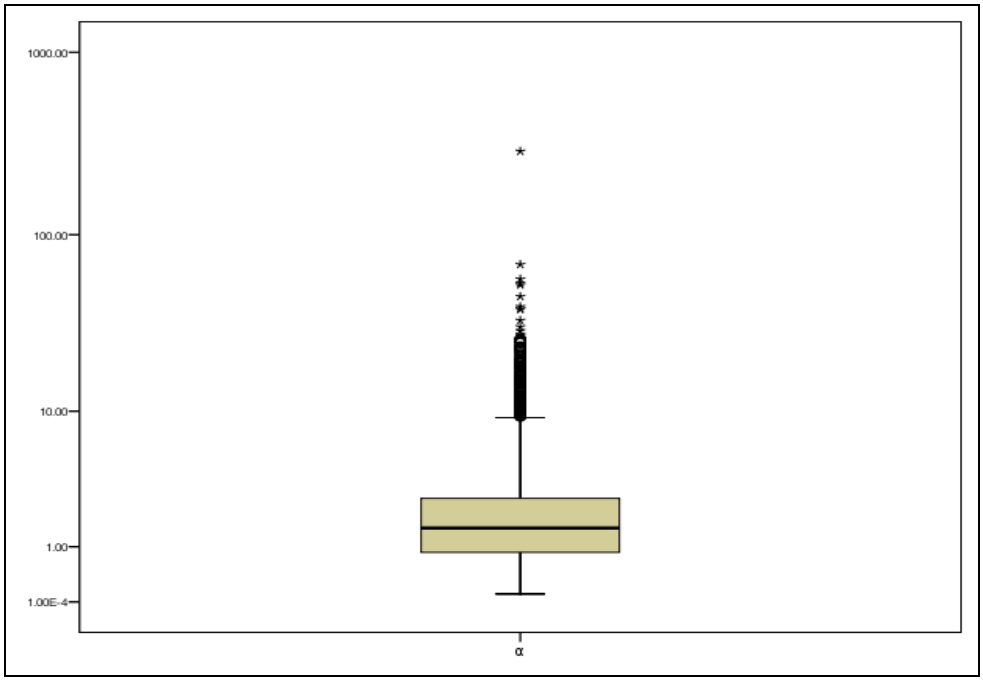


Figure 7.12 Box plot of α for Site A with 30mins STT Showing Numerous Outliers

Table 7.1 Statistics for α

	ECE 11m	Site A 5m	ECE 30m	Site A 30m
Mean	49.87	2.49	35.92	2.50
Mean 95% Confidence (Lower bound)	24.60	2.27	18.43	2.33
Mean 95% Confidence (Upper bound)	75.14	2.71	53.41	2.67
Median	1.04	1.54	1.20	1.53
Variance	867872	33.61	578953	31.95
St.Dev	931.60	5.80	760.89	5.65
Minimum	0.06	0.12	0.06	0.10
Maximum	59851.53	159.08	59851.53	287.50

The figures and table show that data obtained from the proposed dynamic STT model behave similarly to the data obtained from the commonly used 30 minute STT. Hence, further data analysis methods in this study will only explicitly examine the data set generated from the dynamic STT model as it is believed to represent the state of the art in estimating STT.

7.1.2.2 "Wobbles" in the Distribution

During the investigation of the session length per month plots, two interesting observations can be seen.

1. The distributions, at this level of granularity, appear to be stable. Hence, the observable phenomenon seems to repeat in a deterministic fashion.
2. The distributions are not smooth; they include several points of inflection or "wobbles". While it might initially seem reasonable to dismiss these "wobbles" as noise, the fact that they repeat across most of the monthly patterns argues that they are more likely to be signal than noise. This "wobbling" effect has been noted by several other authors investigating related phenomenon (Downey 2001, Ljung and Box 1978, Reed and Jorgensen 2004).

Figures 7.13 and 7.14 display the session length by month graphs for ECE and Site A. For the ECE site, the points of inflection can be seen at approximately 2.6, 3.4 and 3.3. For Site A, two points of inflection happen in quick succession, as can be seen by the smaller graph in Figure 7.14. This figure shows the points of inflection occur at approximately 2.16, 2.18 and 2.5.

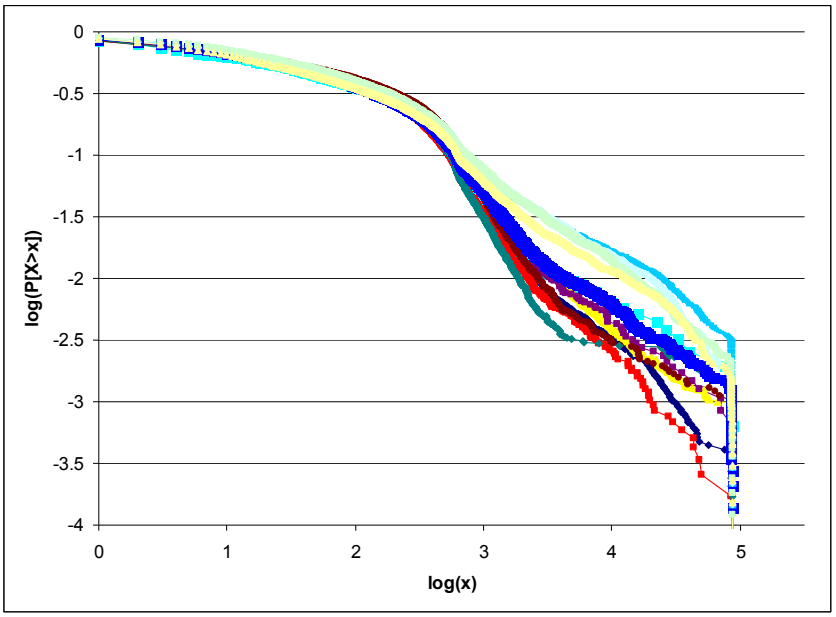


Figure 7.13 "Wobbles" seen in LLCD plots for ECE

This "wobbling" phenomenon argues simple distributions such as Pareto or lognormal distributions cannot be used to model the session workload. Hence, attempting to fit the session length into the Pareto distribution will lead to the wrong conclusion. Various researchers have investigated a range of more complex models to fit this phenomenon:

1. Arlitt et al. (2000, 1998), Barford et al. (1998, 1999) and Downey (2001a, 2001b) have all investigated hybrid models that combine a lognormal distribution with a Pareto tail;
2. Mitzenmacher (2003) investigate, amongst others, a double Pareto distribution;
3. Reed and Jorgensen (2004) investigate a double Pareto-lognormal distribution.

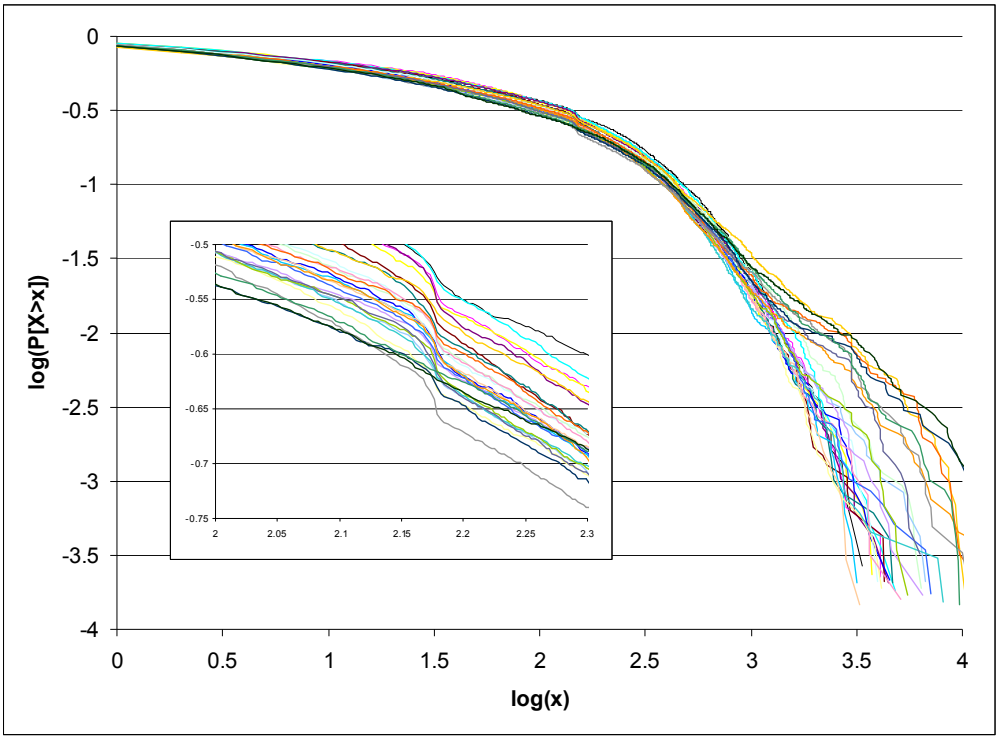


Figure 7.14 "Wobbles" seen in LLCD plots for Site A

While all of these models can provide a superior fit to the "wobbling" phenomenon, there exists no real causal theory that they are an accurate model of the general phenomenon. The alternative argument is that the superior fit is simply the consequence of the greater number of free variables they possess compared to the simpler distributions.

7.1.2.3 Discussions of the Pareto Distribution

Previous studies have demonstrated LLCD plots are not effective at discovering heavy-tailed distributions because of the similarity between the Pareto distribution and the lognormal distribution. Hence, this study will perform an investigation to determine the Pareto distribution's effectiveness at describing the data. Downey (2001, 2005) and Goševa-Popstojanova et al. (2006b) applied the curvature test to explore Pareto and lognormal distribution with conclusions stating that the data can be either Pareto or lognormal. Goševa-Popstojanova et al. (2006b) provides an explanation that the similarity is due to the lack of data at the *far tail*. However, as discussed, the *far tail* of a heavy-tailed distribution will never contain enough data points for any reasonable analysis. Hence, this study will utilize the Q-Q plot (1983) to visually observe the Pareto and lognormal distributions. This is the same approach used by Hernandez-Campos (2004) to investigate Pareto and lognormal distributions. The Q-Q plot allows the quantiles of the data set to be graphically compared against the theoretical distribution (Pareto and lognormal for this investigation). The horizontal axis of the Q-Q plot contains the theoretical quantiles while the vertical axis contains the sorted data values. The natural log-log scale is used because of the possible large values.

The curve generated should follow the 45 degree line if the data quantiles are the same (or very similar) to the theoretical quantile.

Figures 7.15 and 7.16 show the Pareto Q-Q plots for the ECE and Site A sites respectively. Visual observation of these figures shows that the Pareto distribution does not fit extremely well to the data set as the curve does not accurately match the 45 degree line. Further confirmation of this observation can be seen with the detrended Pareto graphs as shown in Figures 7.17 – 7.18. If the plot generated by the detrended graph is not near 0 on the x-axis, then the data set is unlikely to be a good match for the distribution. Once again, these figures show that the observed values deviate from the Pareto distribution very quickly.

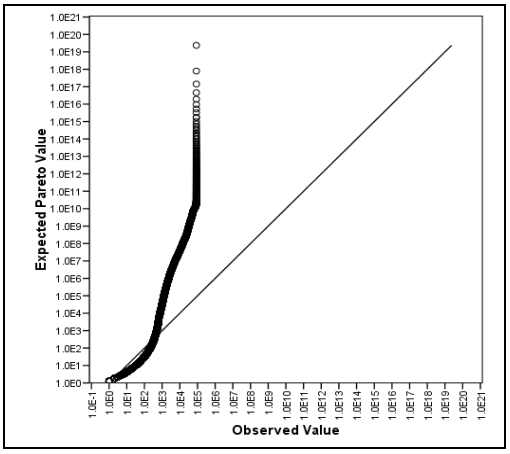


Figure 7.15 Pareto Q-Q Plot for ECE showing the observed values are not near the expected values

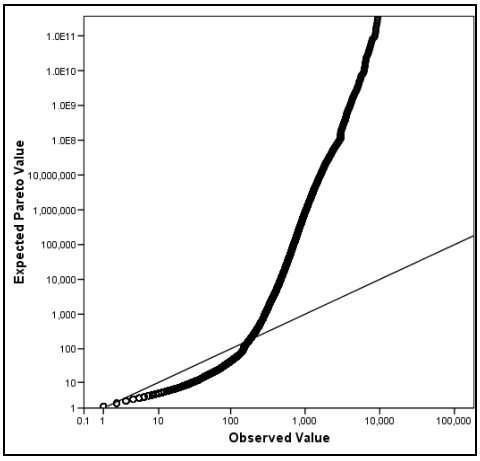


Figure 3.16 Pareto Q-Q Plot for Site A showing the observed values are not near the expected values

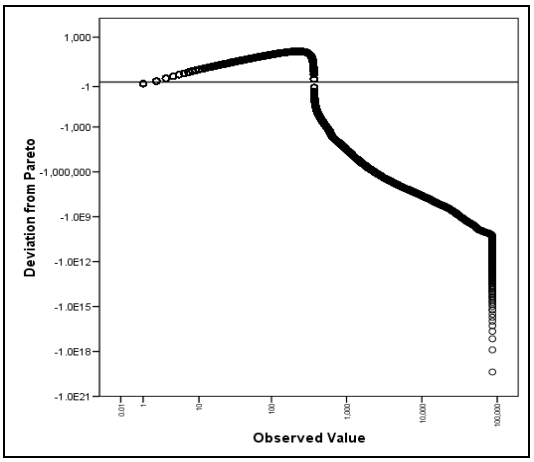


Figure 7.17 Detrended Pareto for ECE showing extreme deviations from the line in the Q-Q plot

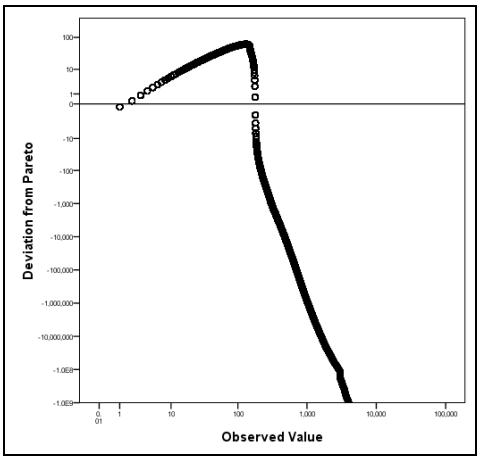


Figure 7.18 Detrended Pareto for Site A showing the observed values are not near the expected values

The lognormal Q-Q plots and detrended plots for the ECE and Site A sites can be seen in Figures 7.19 - 7.22. These figures show that the lognormal distribution also does not describe the distribution of the data accurately.

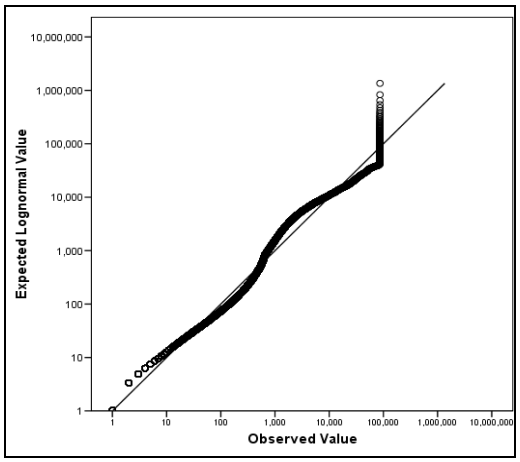


Figure 3.19 Lognormal Q-Q for ECE showing the observed values are not near the expected values

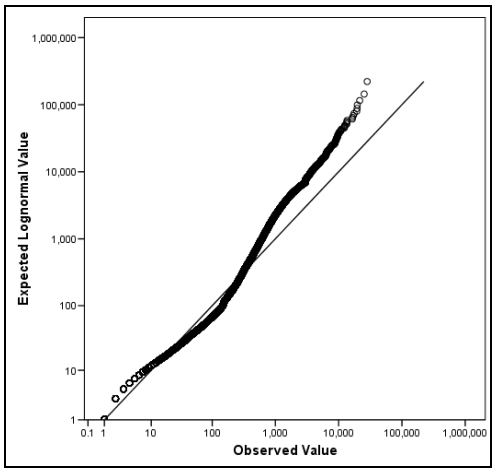


Figure 3.20 Lognormal Q-Q for Site A showing the observed values are not near the expected values

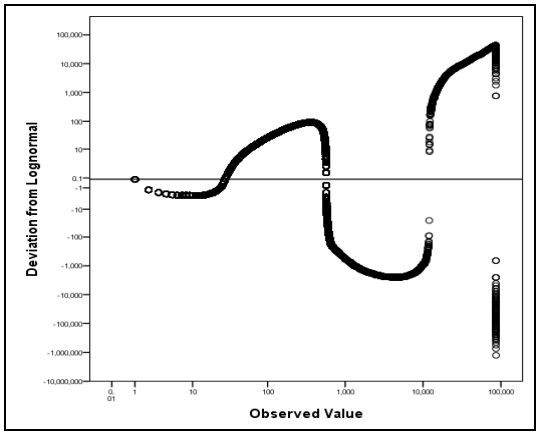


Figure 7.21 Detrended lognormal for ECE showing extreme deviations from the line in the Q-Q plot

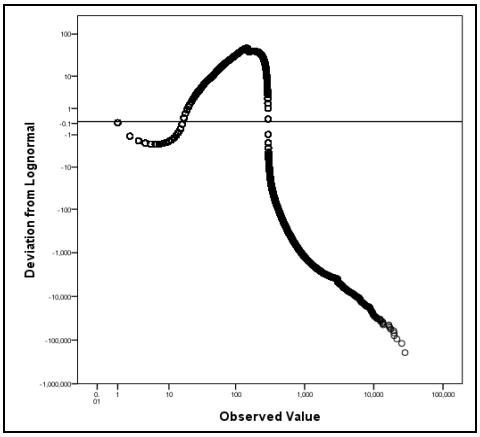


Figure 7.22 Detrended lognormal for Site A showing extreme deviations from the line in the Q-Q plot

To further examine the deviations in the detrended graphs between the lognormal and Pareto distributions, a statistical significance test (t-test) was conducted. The results are presented in Table 7.2. Based on the results, the null hypothesis that the Pareto distribution has a smaller mean (closer to 0) can be rejected. Hence, the alternative hypothesis, which is the mean for the lognormal distribution is "closer" to 0 than the Pareto distribution, can be accepted.

Table 7.2 t-Test to compare the lognormal distribution versus the Pareto distribution

	Site A		ECE	
	lognormal	Pareto	lognormal	Pareto
Mean	95795.9	-3.5×10^{8}	-4299959.0	-3.5×10^{20}
t Stat	9.3		9.9	
t Critical	2.0		2.0	
$P(T \leq t)$	5.0×10^{-11}		4.5×10^{-12}	

However, one can argue that the Pareto distribution is only applied to the tail of the distribution, making a formal analysis difficult to generate due to the lack of definition of the range of the tail. Hence, using this approach, no evidence is found to validate that a Pareto distribution is superior to a lognormal distribution in terms of fitting the underlying data. This observation is consistent with the findings of Downey (2001, 2005) and Goševa-Popstojanova et al. (2006b).

7.1.3 Discussions of the Hill Estimator Results

Using the technique discussed, which is also utilized by Goševa-Popstojanova et al. (2006a, 2006b), the Hill plots for *k* was created for both sites. Goševa-Popstojanova et al. (2006a, 2006b) used 10% and 14% of the upper tail in their Hill plot because *k* appears to settle to a constant value after those points. However, the Hill plots in this study will be displayed across the entire tail to better display the stability of *k*. The Hill estimator can only be performed on the tail of the distribution. Hence, the tail was estimated using the method Goševa-Popstojanova et al. (2006a, 2006b) proposed – even though Section 7.1.2 shows that this approach is not accurate. In order to examine the Hill plot's behavior , a smaller range (0-5) is used for the y-axis as shown in Figures 3.23 – 3.24. These figures show that again α does not stabilize in any part of the graph. In fact, it decreases as the *k* value is increased. There does not appear to be a cut-off point as stated by Goševa-Popstojanova et al. (2006a, 2006b). The Hill plot results further confirms that the heavy-tailed property of the session length may not be an accurate model over the web sites under investigation.

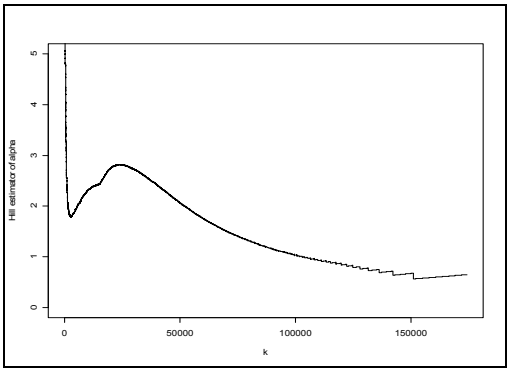


Figure 3.25. Hill estimator for ECE at a smaller range for the y-axis

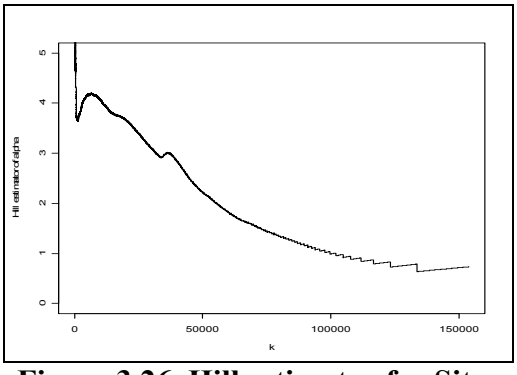


Figure 3.26. Hill estimator for Site A at a smaller range for the y-axis

7.2 Results Discussion

The results from this study show that the session length data may not fit a heavy-tailed distribution. The findings do not confirm the results discovered by Goševa-Popstojanova et al. (2006a, 2006b). However, it should be noted that the websites used in this study have different properties than the websites used in the previous study. Chapter 6 shows that the durations of the log files used in Goševa-Popstojanova et al. (2006a, 2006b) are short. This study performs the investigation over a much longer period of time. Furthermore, although Goševa-Popstojanova et al. (2006a, 2006b) examined a commercial website, the duration is also very short (2 weeks and 1 week), whereas this study examined the commercial website for a 27 month period.

Besides the difference in the duration of the log files, the traffic intensity between the websites in this study and Goševa-Popstojanova et al. (2006a, 2006b) are also vastly different. The websites investigated by Goševa-Popstojanova et al. (2006a, 2006b) have a much heavier traffic load than this study. The busiest website for Goševa-Popstojanova et al. (2006a) received 37.9 millions hits and transferred 97 GB of data during a 3 week period. Goševa-Popstojanova et al. (2006b)'s busiest website received 15.8 million hits and transferred 34.5 GB of data. This study's busiest website, which is ECE, received approximately 2.4 million hits and transferred 22.6 GB of data. The difference in traffic intensity is another possible cause for the different results obtained in this study.

This page is intentionally left blank.

Chapter 8 – Conclusions and Future Works

This dissertation explores two areas of web engineering. The next two sections will present the conclusions and future works for these two areas.

8.1 Web Application Security

In Chapter 2, a research goal of determining whether web application vulnerabilities have any common properties was raised. To reach this goal, four questions were examined.

1. What proportion of security vulnerabilities in web applications can be considered as implementation vulnerabilities? Section 2.3.1 shows that the majority of the known vulnerabilities are of this type. This means researchers should continue to concentrate on implementation vulnerabilities as it will have the most impact on the security of web applications.
2. Are these vulnerabilities the result of interactions between web applications and external systems? The results from Section 2.3.2 show that dynamically created strings passed to functions that allow interactions between the application and an external system cause nearly all of the vulnerabilities in this survey. Hence, developers should be careful when allowing data to flow between the web application and other systems.
3. What is the proportion of vulnerable LOC within a web application? That is, what is the vulnerability density? The results show that the percentage of vulnerable LOC for a web application is extremely small; therefore it can be beneficial to introduce a solution to solve implementation vulnerabilities by concentrating on the CGs with vulnerable LOC.
4. Are implementation vulnerabilities caused by implicit or explicit data flows? Tables 2.9 and 2.10, from Section 2.3.4, show that implementation vulnerabilities for web applications are not caused by implicit data flows. This means efforts on eliminating implementation vulnerabilities can focus on explicit data flows.

Using the results from Chapter 2, Chapter 3 introduces a novel technique aimed at detecting EIVs. The technique contains 4 steps:

1. Sitemap generation
2. Input identification
3. Contamination Data Graph generation
4. Test case selection and execution.

The steps are semi-automated using a web crawler, WAIC, WAGG and a capture playback tool. This divide and conquer approach allocates the repetitive and time consuming steps to various tools, and hence, reduces a significant effort required by security practitioners. Furthermore, security practitioners' expertise and experience remain an essential part which allows the approach to have a high detection rate without any false positives. This approach proposed satisfies the research problems identified in Section 3.2; because it is a software development process, it is applicable to all web applications, the large configuration space and

language dependent. A case study was performed to determine the effectiveness of the approach; it demonstrates that the approach is practical and applicable to commercial strength applications.

EIV analysis has been found to have a very high detection rate for EIVs. The approach found all of the vulnerabilities found by the professionals during a security review; and in addition, found 7 new vulnerabilities missed by the review process. These vulnerabilities were missed because the review process did not correctly identify all of the inputs and hence, several tainted paths were missed. Furthermore, EIV analysis reduced the time required for a security review by 69%.

Because EIV analysis does not allow web administrators to protect their existing platform, Chapter 4 presents an A-NIDS called AIWAS which specifically targets web applications. AIWAS automatically classifies user behaviours into benign or malicious and prevents malicious user behaviours from reaching the web applications under protection.

AIWAS is a learning-based system comprised of two distinct components: the Sentinel and the Oracle. The Sentinel maps the HTTP requests into an IM while preserving important information that aids with the classification of the IM. The Oracle classifies whether the IM is malicious or benign through the use of any supervised ML algorithm.

The results from the case study show that AIWAS is highly capable of classifying IMs in both a 10-fold cross validation test and against real attack test from known vulnerabilities. Additionally, the results demonstrate that the two aggregate algorithms are very consistent with their effectiveness at identifying malicious IMs; hence, they should be the default ML algorithms for AIWAS.

Future works for AIWIAS include modifying AIWAS so it can monitor HTTP requests at a finer granularity. Currently, AIWAS is trained to examine HTTP requests at the web application level. However, each resource within the web application has its own specific usage patterns. For example, a “login” resource will have different usage patterns than a “post comment” resource because the user usually only login once per session while that same user can make multiple “post comments”. Furthermore, as stated 4.1.1 each resource is often associated with different parameters and parameter values. Therefore, by training AIWAS with usage patterns at the resource level, the accuracy of AIWAS can be increased further.

8.2 Data Mining Web Server Logs

Chapter 5 investigates the validity of evaluating web site reliability based on information extracted from existing web server logs. The investigation is a partial follow up to a previously conducted study (Tian et al. 2004). Two additional websites were examined using the methodology proposed in the original study.

The log data for ECE contain two months of data that are one year apart. The log data for the second website (Site A) cover a continuous 15 months of operation. These two websites belong to two organizations that have different reliability requirements for their websites. In this chapter, several findings were discovered:

- Error codes such as 401, 403, and 404 error codes can be divided into different types. Based on the classification of the error types, it is discovered that most errors are no longer source content failures, but are caused by external factors that cannot be controlled by website administrators and content providers. These external factors can be divided into two distinct categories.
- There are issues that exist with the workload information extraction process. The original study explained the difficulties with extracting the byte count workload. However, unique challenges also exist with the extraction of the user and session and hit count workloads. For example each IP may be shared by many users, thus counting each unique IP address as a user will lead to the situation where the counted number of users is actually less than the number of actual users.
- The number of high "value" errors is very low. Hence, the other workloads examined cannot provide better granularity than the daily error rate.
- The Nelson model, used for calculating reliability, is not applicable to some workloads without modifications. The MTBF for a website can be estimated because the total service time can be calculated from the total number of sessions. However, the MTBF will vary depending on the error codes used in the analysis. Thus, the correct error codes need to be selected before reliability evaluation is performed.
- Some of the error codes in response to requests are very similar to requests containing malicious payloads. For example, the 414 error is returned when the URI is too long. A benign client can generate a long URI due to some bug in its code; however the URI can also be too long when an attacker is trying to embed a large piece of JavaScript code to take advantage of a cross-site scripting vulnerability.

Future works for this chapter consist of detailed examination of the user and session workloads. In particular, the investigation will focus on the intra/inter-session characteristics as defined by Goseva-Popstojanova (2006a) in order to examine the behaviors of new users (or sessions) versus repeat users (or sessions) and how these behaviors may affect the reliability of the web server.

In Chapter 6, a new model for estimating the STT is proposed. Having a more accurate STT will allow the session workload to be estimated more accurately which benefits the reliability estimation method explored in Chapter 5. Through empirical observations, this chapter introduces a model that enables web administrators to obtain an accurate STT value for their website. The chapter shows that using a single STT for all websites is not feasible due to the different user profiles associated with each website. The model is then applied to two websites. The first website is from a commercial website that is critical to the

company's operation. The second website is an academic website that serves as a cross reference of the results.

The model, when applied at different resolutions, shows that the results are similar with the STT being shifted depending on the workload and intensity of the resolution. That is, the resulting graphs all have very similar shapes. The results show that the model, while applicable, should only be used at the correct resolution due to different workload behaviours and intensities. Web administrators looking to study the session workload unit should obtain all possible requirements for the study before deciding the correct resolution for applying the model. For example, a study examining user behaviours per month should apply the model at the month resolution. For a long term overall trend, the month resolution can also be used, however, the STT should be recalculated if there are any modifications that can affect the website's usability

Although an absolute correct value for the STT cannot be guaranteed, it is believed that the STT can be estimated more accurately with the proposed model. The model, while simple, allows researchers and web administrators a more dynamic method to obtain STTs that specifically target the websites that they are investigating. Having a more accurate STT is beneficial because it allows more accurate data to be mined while also improving the security and usability of a website. For example, if an online banking website uses an extremely short STT then the user is constantly required to login which decreases the site's usability. However, if it uses an extremely long STT then the risk of the user's account being compromised is increased due to attacks such as XRF (Shiflett 2008), XSS, etc. Furthermore, having an inaccurate STT can lead to session addition (division of a session into two new ones) and subtraction (combining two short sessions into a long one) errors as discussed by Huntington et al. (2008).

The research in Chapter 6 reveals that the distributional properties of the session workload unit are poorly understood. Hence, Chapter 7 examines claims that session length data are sampled from a heavy-tailed distribution. The dependency of the data, the LLCD plot of the data, a Q-Q plot comparison of the performance of the Pareto distribution against the lognormal distribution in fitting the data, and a Hill estimator approach to estimating the tail index of the distribution are all examined in detail. The investigation shows that the data may be dependent; however, the results are disputable because the formulation cannot be extended to cover all possible cases. Furthermore, this chapter confirms that LLCD plots may not be ideal for investigating the heavy-tailed property of session data. The α obtained from the LLCD plot does not stabilize during any part of the tail. Additionally, the Pareto distribution itself is not sufficient for modeling heavy-tailed data because of the "wobble" effect as demonstrated. The Hill estimator was examined and was shown that it also does not provide a stable α value. In fact, α does not stabilize for any k. Finally, the Q-Q plot suggests that the lognormal is a "better" description of the entire distribution, although it cannot be

ruled out that a heavy-tailed distribution may be an adequate distribution of the tail of the distribution due to the imprecise definition of the tail.

Although the investigation in this chapter provides empirical evidence that the session data may not be heavy-tailed, the results can be disputed. The methods utilized, while popular and well known are not entirely accurate. However, no better alternatives exist; until accurate alternative approaches are presented, the heavy-tailed status of the session data is unknown. Therefore future research should consider the matter as being unresolved and should still consider producing short-tailed models to describe this phenomenon.

This page is intentionally left blank.

Bibliography

[Anderson 1972] Anderson J. P., Computer security technology planning study, Technical Report ESD-TR-73-51, United States Air Force, Electronic Systems Division, 1972.

[Antunes and Vieira 2009] Antunes N., Vieira M., Detecting SQL Injection Vulnerabilities in Web Services, Fourth Latin-American Symposium on Dependable Computing, pp. 17-24, 2009.

[Agrawal and Horgan 1990] Agrawal H., Horgan J.R., Dynamic program slicing, Proceedings of the ACM SIGPLAN'90 Conference on Programming Language Design and Implementation, New York, USA, pp. 246–256, 1990.

[Alagar and Ormandjieva 2002] Alagar V.S., Ormandjieva O., Reliability Assessment of Web Applications, 26th Annual International Computer Software and Applications Conference, pp. 405- 412, 2002.

[Alhazmi et al. 2007] Alhazmi O. H., Malaiya Y. K., Ray I., Measuring, Analyzing and Predicting Security Vulnerabilities in Software Systems, Computers and Security Journal, 26(3), pp. 219-228, 2007.

[Alonso et al. 2003] Alonso G., Casati F., Kuno H., Machiraju V., Web Services: Concepts, Architectures, and Applications. Springer Verlag, 2003.

[Arlitt and Jin 2000] Arlitt M., Jin T., A workload characterization study of the 1998 World Cup Web site, IEEE Network, 14(3), pp. 30-37, 2000.

[Arlitt and Williamson 1997] Arlitt M.F., Williamson C.L., Internet Web servers: workload characterization and performance implications. IEEE/ACM Transactions on Networking, 5(5), pp. 631-645, Oct. 1997.

[Arlitt et al. 1998] Arlitt M., Friedrich R., Jin T., Workload characterization of a Web proxy in a cable modem environment, ACM Sigmetrics Performance Evaluation Review, 27(2), pp. 25 – 36, 1998.

[Arlitt et al. 2001] Arlitt M.F., Krishnamurthy D., Rolia J., Characterizing the scalability of a large web-based shopping system, ACM Transactions on Internet Technology 1(1), pp.44-69, 2001.

[Baldi et al. 2000] Baldi P., Brunak S., Chauvin Y., C.A.F., Andersen, H. Nielsen, Assessing the accuracy of prediction algorithms for classification: an overview, Bioinformatics, 16(5), pp.412-424, 2000.

[Balzarotti et al. 2007] Balzarotti D., Cova M., Felmetsger V.V., Vigna G., Multi-Module Vulnerability Analysis of Web-based Applications, Proc. 14th ACM Conference on Computer and Communication Security, pp. 25-35, 2007.

[Bandhakavi et al. 2007] Bandhakavi S., Bisht P., Madhusudan P., Venkatakrishman V., CANDID: Preventing SQL Injection Attacks Using Dynamic Candidate Evaluations, Proceedings of the 14th ACM conference on Computer and communications security, Virginia, USA, pp. 12-24, 2007.

[Barford and Crovella 1998] Barford P., Crovella M. E., Generating representative Web workloads for network and server performance evaluation, Performance SIGMETRICS '98, pp. 151-160, 1998.

[Barford et al. 1999] Barford P., Bestavros A., Bradley A., Crovella M., Changes in Web client access patterns: Characteristics and caching implications. World Wide Web: Special Issue on Characterization and Performance Evaluation, Vol. 2, pp. 15-28, 1999.

[Basili et al. 1994] Basili V., Caldeira G. Rombach H. D., The Goal Question Metric Approach, Encyclopedia of Software Engineering, Wiley & Sons, 1994.

[Baskerville and Pries-Heje 2004] Baskerville, R., Pries-Heje, J., Short cycle time systems development, Information Systems Journal, 14(3), pp. 237-264, 2004.

[Batista et al. 2004] Batista G.E.A.P.A., Prati R.C., Monard M.C., A study of the behavior of several methods for balancing machine learning training data, ACM SIGKDD Explorations Newsletter, 6(1), pp. 20-29, 2004.

[Beizer 1995] Beizer B., Black-Box Testing: Techniques for Functional Testing of Software and Systems, Wiley: New York, 1995.

[Berendt et al. 2001] Berendt B., Mobasher B., Spiliopoulou M., Wiltshire J., Measuring the accuracy of sesionizers for web usage analysis, Proceedings of the workshop on web mining at the first SIAM international conference on data mining, pp. 7-14, 2001.

[Boehm and Abts 1999] Boehm B., Abts C., COTS Integration: Plug and Pray?, IEEE Computer, 32 (1): pp. 135-138, January 1999.
[Bolzoni and Etalle 2008] Bolzoni D., Etalle S., Boosting Web Intrusion Detection Systems by Inferring Positive Signatures, On the Move to Meaningful Internet Systems, Vol. 5332, pp. 938-955, 2008.
[Boyd and Keromytis 2004] Boyd S. Keromytis A., SQLrand: Preventing SQL injection attacks, 2nd Applied Cryptography and Network Security (ACNS) Conference, pp. 292-304, 2004.
[Breiman 2001] Breiman L., Random Forests, Machine Learning, 45(1) pp. 5-32, 2001.
[Brockwell and Davis 1991] Brockwell P.; Davis R., Time Series: theory and Methods, Springer-Verlag, 1991.
[Buehrer et al. 2005] Buehrer G.T., Weide B.W., Sivilotti P.A.G., Using Parse Tree Validation to Prevent SQL Injection Attacks, In Proc. of the 5th Intl. Workshop on Software Engineering and Middleware (SEM '05), Lisbon, Portugal, pp. 106–113, 2005.
[Catledge and Pitkow 1995] Catledge L.D., Pitkow J.E., Characterizing browsing strategies in the World-Wide Web, Proceedings of the Third International World-Wide Web conference on Technology, tools and applications, pp.1065-1073, 1995.
[Chaudhri et al. 2003] Chaudhri A., Zicari R., Rashid A., XML Data Management: Native XML and XML Enabled DataBase Systems, USA: Addison-Wesley, 2003.
[Chawla et al. 2002] Chawla N.V., Bowyer K.W., Hall L.O., Kegelmeyer W.P., SMOTE: synthetic minority over-sampling technique, Journal of Artificial Intelligence Research, 16(1), pp.321-357, 2002.
[Chawla et al. 2004] Chawla N.V., Japkowicz N., Kotcz A., Editorial: special issue on learning from imbalanced data sets, ACM SIGKDD Explorations Newsletter, 6(1), pp. 1-6, 2004.
[Chen 2002]Chen, Y-T., On the Robustness of Ljung-Box and McLeod-Li Q tests: a simulation study, Economics Bulletin, 3(17), pp. 1–10, 2002.
[Chen et al. 2003] Chen Z., Fowler R.H., Fu A.W.-C., Linear time algorithms for finding maximal forward references, Information Technology: Coding and Computing [Computers and Communications], 2003. Proceedings. ITCC 2003. International Conference on, Vol. 28(30), pp. 160- 164, 2003.
[Cheng et al. 2008] Cheng YC, Laih CS, Lai GH, Chen CM, Chen T, Defending On-Line Web Application Security with User-Behaviour Surveillance, Third International Conference on Availability, Reliability and Security, pp. 410-415, 2008.
[Cherkasova and Phaal 2002] Cherkasova L., Phaal P., Session-Based Admission Control: A Mechanism for Peak Load Management of Commercial Web Sites, Transactions on Computers, 51(6), pp. 669-685, 2002.
[Cooley at al. 1997] Cooley R., Mobasher B., and Srivastava J., Web mining: Information and pattern discovery on the World Wide Web. In International Conference on Tools with Artificial Intelligence, pp. 558–567, 1997.
[Cooley et al. 1999] Cooley R., Mobasher B., and Srivastava J., Data preparation for mining world wide web browsing patterns. Knowledge and Information Systems, Vol. 1(1), pp. 5 – 32, 1999.
[Cova et al. 2007a] Cova M., Balzorotti D., Felmetsger V., Vigna G., Swaddler: An Approach for the Anomaly-Based Detection of State Violations in Web Applications, Recent Advances in Intrusion Detection, pp. 63-86, 2007.
[Cova et al. 2007b] Cova M., Felmetsger V., Vigna G., Vulnerability Analysis of Web-based Applications, Test and Analysis of Web Services, pp. 363-394, 2007.
[Cowan et al. 1998] Cowan C., Pu C., Maier D., Hinton H., Bakke P., Beattie S., Grier A., Wagle P., and Zhang Q., StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks, 7th USENIX Security Conference, San Antonio TX, pp. 63-78, 1998.
[Cremonesi and Serazzi 2002] Cremonesi P., Serazzi G., End-to-End Performance of Web Services, Performance Evaluation of Complex Systems: Techniques and Tools, Performance 2002 Tutorial Lectures, Lecture Notes in Computer Sciences, Vol. 2459, pp. 158-178, 2002.
[Crovella and Bestavros 1997] Crovella M.E., Bestavros A., Self-Similarity in Word Wide Web Traffic: Evidence and Possible Causes, IEEE/ACM Transactions on Networking, 5(6), pp. 835 – 846, 1997.

[Curbera et al. 2002] Curbera F., Duftler M., Khalaf R., Nagy W., Mukhi N., Weerawarana S., Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing, 6(2), pp.86-93, March 2002.
[Davis and Resnick 1985] Davis R.; Resnick S., Limit theory for the sample covariance and correlation functions of moving averages, Annuals of Statistics, Vol. 13, pp. 179 – 195, 1985.
[Denning and Denning 1997] Denning D.E., Denning P.J., Certification of programs for secure information flow, Comm. Of the ACM, New York, USA, ACM, pp. 504-513, 1997.
[Dhamija et al. 2006] Dhamija R., Tygar J. D., Hearst M., Why phishing works, Proceedings of the SIGCHI conference on Human Factors in computing systems, Montréal, Québec, Canada, April 22-27, 2006.
[Doar 2005] Doar M.B., Practical Development Environments, O'Reilly Media, 2005.
[Downey 2001a] Downey A.B., Evidence for Long-tailed distributions in the Internet, Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement, pp. 229 – 241, 2001.
[Downey 2001b] Downey A.B., The structural cause of fie size distributions, Proceedings of the IEE/ACM International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 361 – 370, 2001.
[Downey 2005] Downey A.B., Lognormal and Pareto Distributions in the Internet, Computer Communications, 28(7), pp. 790-801, 2005.
[Eaton and Memon 2007] Eaton C., Memon A.M., An Empirical Approach to Testing Web Applications Across Diverse Client Platform Configurations, International Journal on Web Engineering and Technology (IJWET), Special Issue on Empirical Studies in Web Engineering, 2007.
[Eirinaki and Vazirgiannis 2003] Eirinaki M., Vazirgiannis M., Web mining for web personalization, ACM Transactions on Internet Technology, 3(1), pp.1-27, 2003.
[Estevez-Tapiador et al. 2005] Estevez-Tapiador JM, Garcia-Teodoro P, Diaz-Verdejo JE, Detection of web-based attacks through Markovian protocol parsing, ISCC05, pp. 457-462, 2005.
[Evans and Larochelle 2002] Evans D., Larochelle D., Improving Security Using Extensible Lightweight Static Analysis, IEEE Software, pp. 42-51, 2002.
[Fawcett 2003] Fawcett T., ROC graphs: notes and practical considerations for researchers, Technical Report, HP Laboratories, 2003.
[Feigen and Resnick 1999] Feigen P.D.; Resnick S.I., Pitfalls of fitting autoregressive models for heavy-tailed time series, Extremes, 1(4), pp. 391–422, 1999.
[Figueiredo et al. 2005] Figueiredo D.R., Jiu B., Feldmann A., Misra V., Towsley D., Willinger W., On TCP and self-similar traffic, Performance Evaluation, Vol. 61, pp. 129–141, 2005.
[Fisher 1983] Fisher N.I., Graphical Methods in Nonparametric Statistics: A Review and Annotated Bibliography, International Statistical Review, Vol. 51, pp. 25-58, 1983.
[Fleiss 1975] Fleiss J.L., Measuring Agreement between Two Judges on the Presence or Absence of a Trait, Biometrics, 31(3), pp. 651-659, 1975.
[Forrest et al. 1996] Forrest S., Hofmeyr S, Somayaji A., Longstaff T., A sense of self for Unix processes. Proceedings of IEEE Symposium on Security and Privacy, pp. 120-128, 1996.
[Frankl and Weyuker 1988] Frankl P.G., Weyuker E.J., An applicable family of data flow testing criteria, IEEE Transactions on Software Engineering, 14(10), pp.1483-1498, Oct 1988.
[Fu et al. 1999] Fu Y., Sandhu K., Shih M., Clustering of web users based on access patterns, International Workshop on Web Usage Analysis and User Profiling (WEBKDD'99), San Diego, CA, 1999.
[Gabaix 1999] Gabaix X., Zipf's law for cities: an explanation, Quarterly Journal of Economics, 114(3), pp. 739–767, 1999.
[Galleta et al. 2004] Galletta D.F., Henry R., McCoy S., Polak P., Web site delays: How tolerant are users?, Journal of the AIS, 5(1): pp. 1-28, 2004.
[Garret 2008] Garrett J.J., Ajax: A new approach to web applications, http://www.adaptivepath.com/publications/essays/archives/000385.php, last accessed January 20, 2008..
[Goldstein et al. 2004] Goldstein M.L., Morris S.A., Yen G.G., Problems with fitting to the power-law distribution, European Physics Journal B, Vol. 41, pp. 255- 258, 2004.

[Gong et al. 2001] Gong W., Liu Y., Misra V., Towsley D., On the tails of web file size distributions, Proceedings of the 39th Allerton Conference on Communication, Control and Computing, 2001.
[Goševa-Popstojanova et al. 2004] Goševa-Popstojanova K., Mazimdar S., and Singh A., Empirical Study of Session-based Workload and Reliability for Web Servers, 15th IEEE International Symposium on Software Reliability, pp. 403-414, 2004.
[Goševa-Popstojanova et al. 2006a] Goševa-Popstojanova K., Singh A.D., Mazimdar S., Li F., Empirical Characterization of Session–Based Workload and Reliability for Web Servers, Empirical Software Engineering, Springer Netherlands, 11(1), pp. 71-117, 2006.
[Goševa-Popstojanova et al. 2006b] Goševa-Popstojanova K., Li F., Wang X., Sangle A., A Contribution Towards Solving the Web Workload Puzzle, International Conference on Dependable Systems and Networks (DSN'06), pp. 505-516, 2006.
[Granger 2003] Granger, S., Social Engineering Fundamentals, Part I: Hacker Tactics, Security Focus, http://www.securityfocus.com/infocus/1527, 2003.
[Halfond and Orso 2005] Halfond W. G., Orso A., AMNESIA: Analysis and Monitoring for NEutralizing SQL-Injection Attacks, In Proceedings of 20th ACM International Conference on Automated Software Engineering (ASE), Long Beach, CA, USA, pp. 174-183, 2005.
[Halfond et al. 2006] Halfond W.G., Orso A., Manolios P., Using positive tainting and syntax-aware evaluation to counter SQL injection attacks, In Proceedings of the 14th ACM SIGSOFT international Symposium on Foundations of Software Engineering, Portland, Oregon, USA, pp. 175-185, 2006.
[Halfond et al. 2008] Halfond W.G.J., Orso A., Manolios P., WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation, IEEE Transactions on Software Engineering, 34(1), pp. 65-81, 2008.
[Harrold and Rothermel 1994] Harrold M.J., Rothermel G., Performing data flow testing on classes, In Proceedings of the 2nd ACM SIGSOFT Symposium on Foundations of Software Engineering (New Orleans, Louisiana, United States, December 06 - 09, 1994), SIGSOFT '94, ACM Press, New York, NY, pp. 154-163, 1994.
[He and Goker 2000] He D., Goker A., Detecting session boundaries from Web user logs, Proceedings of the 22nd Annual Colloquium on Information Retrieval Research, pp.57-66, British Computer Society, 2000.
[Heberlein et al. 1990] Heberlein L.T., Dias G.V., Levitt K.N., Mukherjee B., Wood J., Wolber D., A network security monitor, Proceedings of IEEE Symposium on Security and Privacy, pp. 296-304, 1990.
[Hernández-Campos et al. 2004] Hernández-Campos F., Marron J. S., Samorodnitsky G., Smith F. D., Variable heavy tails in Internet traffic, Performance Evaluation, 58(2+3), pp. 261-284, 2004.
[Heydon and Najork 1999] Heydon A., Najork M., Mercator: A scalable, extensible Web crawler, World Wide Web, 2(4), pp. 219–229, December 1999.
[Hill 1975] Hill B., A simple approach to inference about the tail of a distribution, Annuals of Statistics, Vol. 3, pp. 1163–1774, 1975.
[Howard and LeBlanc 2003] Howard M., LeBlanc D., Writing Secure Code, Second Edition, Microsoft Press, 2003.
[Howden 1975] Howden W. E., Methodology for the generation of program test data, IEEE Trans. Comput., C-24(5), pp. 554-559, May 1975.
[Huang et al. 2003] Huang Y.W., Huang S.K., Lin T.P., Tsai C.H., Web application security assessment by fault injection and behavior monitoring, 12th International Conference on World Wide Web, pp. 148-159, 2003.
[Huang et al. 2004] Huang Y. W., Yu F., Hang C., Tsai C.H., Lee D.T., Kuo S. Y., Securing web application code by static analysis and runtime protection, in WWW '04: Proceedings of the 13th International Conference on World Wide Web. New York, NY, USA: ACM Press, pp. 40-52, 2004.
[Huang et al. 2004] Huang X., Peng F., An A., Schuumans D., Dynamic web log session identification with statistical language models, Journal of the American Society for Information Science and Technology, Vol. 55, pp. 1290-1303, 2004.

[Huntington et al. 2008] Huntington P., Nicholas D., Jamali H.R., Website usage metrics: A re-assessment of session data, Information Processing & Management. Vol. 44, pp. 358-372, 2008.
[Hurst 2004] Hurst A., Analysis of Perl's taint mode, http://hurstdog.org/papers/hurst04taint.pdf, 2004.
[Huynh and Miller 2005] Huynh T., Miller J., Further Investigations into Evaluating Website Reliability, 4th International Symposium on Empirical Software Engineering, pp 162-171, 2005.
[Huynh and Miller 2009] Huynh T., Miller J., Empirical Observations on the Session Timeout Threshold, Journal of Information Processing & Management, 45(5), pp.513-528, 2009.
[Jansen and de Vries 1991] Jansen D.W. de Vries C.G., On the frequency of large stock returns: putting booms and busts into perspective, Review of Economics and Statistics, Vol. 73, pp. 18–24, 1991.
[Jansen and Spink 2003] Jansen B.J., Spink A., An Analysis of Web Documents Retrieved and Viewed, The 4th International Conference on Internet Computing, pp.65-69, 2003.
[John and Langley 1995] John G.H., Langley P., Estimating Continuous Distributions in Bayesian Classifiers, 11th Conference on Uncertainty in Artificial Intelligence, pp. 338-345, 1995.
[Johnson and Wagner 2004] Johnson R., and Wagner D., Finding user/kernel pointer bugs with type inference, In Proceedings of the 2004 Usenix Security Conference, San Diego, CA, USA, pp. 119–134, 2004.
[Jolliffee 1986] Jolliffee I.T., Principal Component Analysis, Springer-Verlag, New York, 1986.
[Jovanovic et al. 2006] Jovanovic N., Kruegel C., Kirda E., Pixy: A Static Analysis Tool for Detecting Web Application Vulnerabilities, In 2006 IEEE Symposium on Security and Privacy, Berkeley/Oakland, CA, USA, pp. 258-263, 2006.
[Kallepalli and Tian 2001] Kallepalli C., Tian J., Measuring and Modeling Usage and Reliability for Statistical Web Testing, IEEE Trans. Software Eng., 27 (11), pp. 1023-1036, 2001.
[Kals et al. 2006] Kals S., Kirda E., Kruegel C., Jovanovic N., SecuBat: A Web Vulnerability Scanner, The 15th International World Wide Web Conference (WWW 2006), Edinburgh, Scotland, pp. 247-256, 2006.
[Kiezun et al. 2008] Kiezun A., Guo P.J., Jayaraman K., Ernst M.D., Automatic Creation of SQL Injection and Cross-Site Scripting Attacks, Proceedings of the 2009 IEEE 31st International Conference on Software Engineering, Vancouver, British Columbia, Canada, pp. 199-209, 2008.
[Kristol and Montulli 2000] Kristol D.M., Montulli L., HTTP State Management Mechanism, RFC 2965 (http://tools.ietf.org/html/rfc2965), October 2000.
[Krugel et al. 2002] Krugel C., Toth T., Kirda E., Service specific anomaly detection for network intrusion detection, ACM Symposium on Applied Computing, pp. 201-208, 2002.
[Kruegel and Vigna 2003] Kruegel C., Vigna G., Anomaly detection of web-based attacks, ACM conference on Computer and communication security, pp. 251-261, 2003.
[Kruegel and Vigna 2005] Kruegel C., Vigna G., A multi-model approach to the detection of web-based attacks, Computer Networks 48(5), pp. 717-738, 2005.
[Kuncheva and Rodriguez 2007] Kuncheva L.I., Rodriguez J.J., An Experimental Study on Rotation Forest Ensembles, Multiple Classifier Systems, Vol. 4472, pp. 459-468, 2007.
[Landwehr et al. 2005] Landwehr N.L., Hall M., Frank E., Logistic Model Trees, Machine Learnings, 59(1-2), pp. 161-205, 2005.
[Laski and Korel 1983] Laski, J. W., Korel, B., Data flow oriented program testing strategy, IEEE Transactions on Software Engineering, 9(3), pp. 347-354. 1983.
[Lazarevic et al. 2005] Lazarevic A, Kumar V, Srivastava J, Intrusion Detection: A Survey, Managing Cyber Threats, pp.19-78, 2005.
[Lin and Chen 2006] Lin J.-C., Chen J.-M., An Automatic Revised Tool for Anti-Malicious Injection, Sixth IEEE International Conference on Computer and Information Technology (CIT'06), Seoul, South Korea, pp. 164-170, 2006.
[Lipner 2000] Lipner S.B., Security and Source Code Access: Issues and Realities, 2000 IEEE Symposium on Security and Privacy (S&P 2000), pp. 124-125, 2000.
[Lippmann et al. 2000] Lippmann R., Haines J., Fried D., Korba J., Das K., The 1999 DARPA off-line intrusion detection evaluation, Computer Networks 34(4), pp. 579–595, 2000.

[Liu et al. 2009] Liu A., Yuan Y., Wijesekera D., Stavrou A., SQLProb: a proxy-based architecture towards preventing SQL injection attacks, Proceedings of the 2009 ACM symposium on Applied Computing, Honolulu, Hawaii, pp.2054-2061, 2009.
[Liu et al. 2000] Liu C. H., Kung D., Hsia P., and Hsu C. T., Structure testing of web applications, In Proceedings of the 11th Annual International Symposium on Software Reliability Engineering, pp. 84–96, San Jose CA, October 2000.
[Livshits and Lam 2005] Livshits V.B., Lam M.S., Finding Security Vulnerabilities in Java Applications with Static Analysis, In Proceedings of the 14th Usenix Security Symposium, Aug. 2005.
[Ljung and Box 1978] Ljung G.M., Box G.E.P., On a measure of lack of fit in time series models, Biometrika 65, pp. 553-564, 1978.
[Lyu 1995] Lyu M.R., Handbook of Software Reliability, McGraw-Hill, 1995.
[Ma and TIan 2003] Ma L., Tian J., Analyzing Errors and Referral Pairs to Characterize Common Problems and Improve Web Reliability, 3rd International Conference on Web Engineering, pp. 314-323., 2003.
[Mahoney and Chan 2004] Mahoney M., Chan P., An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection, Recent Advances in Intrusion Detection (RAID), pp. 220–237, 2004.
[Mahoui and Cunningham 2000] Mahoui M., Cunningham S.J., A comparative transaction log analysis of two computing collections, Lecture Notes in Computer Science. Vol 1923, pp.418-423, 2000.
[Martin et al. 2005] Martin M., Livshits B., and Lam M. S., Finding Application Errors and Security Flaws Using PQL: a Program Query Language, In OOPSLA ’05: Proc. of the 20th Annual ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, San Diego, CA, USA, pp. 365–383, 2005.
[Martin and Lam 2008] Martin M., Lam M., Automatic Generation of XSS and SQL Injection Attacks with Goal-Directed Model Checking, Proceedings of the 17th conference on Security symposium, San Jose, CA, pp. 31-43, 2008.
[Mat-Hassan and Levene 2005] Mat-Hassan M., Levene M., Associating search and navigation behavior through log analysis, Journal of the American Society for Information Science and Technology, 56(9), pp.913-934, 2005.
[McAllister et al. 2008] McAllister S., Kirda E., Kruegel C., Leveraging User Interactions for In-Depth Testing of Web Applications, Recent Advances in Intrusion Detection, vol 5230, pp. 191-210, 2008.
[McGraw 2004] McGraw G., Software Security, IEEE Security & Privacy, 2(2), pp. 80–83, 2004.
[McHugh 2000a] McHugh J., The 1998 Lincoln Laboratory IDS evaluation, Recent Advances in Intrusion Detection (RAID), pp. 145–161, 2000.
[McHugh 2000b] McHugh J., Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory, ACM Transactions on Information Systems Security, 3(4), pp. 262–294, 2000b.
[Menascé et al. 1999] Menascé D. A., Almeida V.A., Fonseca R., Mendes M.A., A methodology for workload characterization of E-commerce sites, In Proceedings of the 1st ACM Conference on Electronic Commerce, ACM Press, pp. 119-128, 1999.
[Menasce et al. 2000a] Menasce D., Almeida V., Foneca R., Mendes M., Business-oriented Resource Management Policies for E-commerce Servers, Performance Evaluation, 32 (2-3), pp. 223-239, 2000.
[Menasce et al. 2000b] Menasce D., Almeida V., Ried R., In Search of Invariants for E-Business Workloads, 2nd ACM Conference on Electronic Commerce, Minneapolis, MI, pp. 56-65, 2000.
[Menascé et al. 2002] Menascé D.A., Barbara, D., Almeida V., Ribeiro F., Fractal characterization of web workloads, 11th International World Wide Web Conference, pp. 7-11, 2002.
[Miller and Bharat 1998] Miller R.C., Bharat K., SPHINX: A framework for creating personal, site-specific Web crawlers, In Proceedings of the Seventh International World Wide Web Conference, pp. 119-130, April 1998.
[Mitzenmacher 2003] Mitzenmacher M., Dynamic Models for File Sizes and Double Pareto Distributions, Internet Mathematics, 1(3), pp. 305–333, 2003.

[Mobasher et al. 2000] Mobasher B., Cooley R., Srivastava J., Automatic personalization based on Web usage mining, Communications of the ACM, 43(8), pp. 142-151, 2000.
[Moody and Palomino 2003] Moody K., Palomino M., SharpSpider: Spidering the Web through Web Services, First Latin American Web Congress (LA-WEB 2003), 2003.
[Montgomery and Faloutsos 2001] Montgomery A.L, Faloutsos C., Identifying web browsing trends and patterns, IEEE Computer, 34(7), pp. 94-95, 2001.
[Musa et al. 1987] Musa J.D., Iannino A., and Okumoto K., Software Reliability: Measurement, Prediction, Application, McGraw-Hill, 1987.
[Musciano and Kennedy 2002] Musciano C., Kennedy B., HTML and XHTML: The Definitive Guide, O'Reilly & Associates, Inc., Sebastopol, CA, 2002.
[Myers 1979] Myers G.J., The Art of Software Testing, Wiley, 1979.
[Nah 2002] Nah F.H., A Study of Web Users' Waiting Time, Intelligent Support Systems Technology: Knowledge Management, Vijayan Sugumaran (editor), IRM Press, pp. 145-152, 2002.
[Needleman and Wunsch 1970] Needleman S.B., Wunsch C.D., A general method applicable to the search for similarities in the amino acid sequence of two proteins, J Mol Biol, 48(3), pp. 443–53, 1970.
[Nelson 1978] Nelson E., Estimating Software Reliability from Test Data, Microelectronics and Reliability, 17(1), pp. 67-73, 1978.
[Nguyen-Tuong et al. 2005] Nguyen-Tuong A., Guarnieri S., Greene D., Shirley J., Evans D., Automatically Hardening Web Applications Using Precise Tainting, In Proceedings of the 20th IFIP International Information Security Conference, Chiba, Japan, pp. 372-382, 2005.
[Nicholas et al. 2000] Nicholas D., Huntington P., Lievesley N., Wasti A., Evaluating consumer Web site logs: Case study The Times/Sunday Times Web site, Journal of Information Science, 26(6), pp. 399-411, 2000.
[Nicholas et al. 2006a] Nicholas D., Huntington P., Jamali H.R., Watkinson A., What deep log analysis tells us about the impact of big deal, case study OhioLink, Journal of Documentation, 62(4), pp. 482-508, 2006.
[Nicholas et al. 2006b] Nicholas D., Huntington P., Jamali H.R., Watkinson A., The information seeking behaviour of the users of digital scholarly journals, Information Processing and Management, 42(5), pp. 1345-1365. 2006.
[Ntafos 1984] Ntafos S.C., Required element testing, IEEE Trans. Software Eng., SE-10(6), pp. 795-803, Nov. 1984.
[Offutt 1994] Offutt A.J., A practical system for mutation testing: Help for the common programmer, Proceedings of the International Test Conference, pp. 824-830, 1994.
[Offutt 2002] Offutt J., Quality Attributes of Web Applications, IEEE Software: Special Issue on Software Engineering of Internet Software, 19 (2), pp. 25-32, 2002.
[Offutt 2004] Offutt J., Wu Y., Du X., Huang H., Bypass testing of Web applications In Proceedings of The Fifteenth IEEE International Symposium on Software Reliability Engineering, Saint-Malo, Bretagne, France, pp.187-197, 2004.
[Ollman 2004] Ollman G., The phishing guide - understanding and preventing phishing attacks, White Paper, Next Generation Security Software Ltd., 2004.
[OWASP 2010] OWASP, Guide to Building Secure Web Applications and Web Services: Data Validation, OWASP, http://www.owasp.org/index.php/Guide_Table_of_Contents, last accessed January 9, 2010.
[Park and Park 2008] Park Y.J., Park J.C., Web Application Intrusion Detection System for Input Validation Attack, Third 2008 International Conference on Convergence and Hybrid Information Technology, pp. 498-504, 2008.
[Pallis et al. 2005] Pallis G., Angelis L., Vakali A., Model-based cluster analysis for web users sessions, 15th International Symposium on Methodologies for Intelligent Systems, Springer-Verlag Berlin Heideberg, pp. 219-227, 2005.
[Pankratz 1983] Pankratz A., Forecasting with univariate Box-Jenkins models: Concepts and cases, New York: John Wiley and Sons, 1983.
[Pietraszek and Berghe 2005] Pietraszek T., Berghe C.V., Defending Against Injection Attacks through Context-Sensitive String Evaluation, In Proceedings of Recent Advances in Intrusion Detection (RAID2005), Seattle, Washington, USA, pp. 124-145, 2005.

[Pitkow 1999] Pitkow J.E., Summary of WWW characterizations, World Wide Web, 2(1-2), pp. 3-13, 1999.
[Provost and Fawcett 2001] Provost F.J., Fawcett T., Robust classification for imprecise environments, Machine Learning, 42(3), pp. 203-231, 2001.
[Raghavan and Garcia-Molina 2001] Raghavan S., Garcia-Molina H., Crawling the hidden web, In Proc. of 27th Int. Conf. on Very Large Databases, pp. 129-138, Sept. 2001.
[Rapid7 2005] Rapid7, Vulnerability Management Trends, Issue 2, pp. 1-9, 2005.
[Rapps and Weyuker 1985] Rapps S., Weyuker E.J., Selecting software test data using data flow information, IEEE Trans. Software Eng., 11(4), pp. 367-375. April 1985.
[Reed et al. 2004[Reed J.W., Jorgensen M., The Double Pareto-Lognormal Distribution—A New Parametric Model for Size Distributions, Communications in Statistics – Theory and Methods, pp. 1733 – 1753, 2004.
[Resnick 1997] Resnick S.I., Heavy Tail modeling and teletraffic data, The Annuals of Statistics, 25(5), pp 1805–1849, 1997.
[Rezaul and Grout 2006] Rezaul K.M., Grout V., A Comparison of Methods for Estimating the Tail Index of Heavy-tailed Internet Traffic, Proceedings of the 2nd International Joint e-Conference on Computer, Information, and Systems Sciences, and Engineering, pp.219-222, 2006.
[Rittel and Webber 1973] Rittel H.W.J., Webber M.M., Dilemmas in a general theory of planning, Policy Sciences, Vol. 4, pp. 155-169, 1973.
[Rodriguez et al. 2006] Rodriguez J.J., Kuncheva L.I., Alonso C.J., Rotation Forest: A new classifier ensemble method, IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(10), pp. 1619-1630, 2006.
[Rose et al. 2001] Rose G.M., Lees J., and Meuter M., A refined view of download time impacts on e-consumer attitudes and patronage intentions toward e-retailers, The International Journal on Media Management, 3(2), pp.105-111, 2001.
[Rosenstein 2000] Rosenstein M., What is Actually Taking Place in Web Sites: E-Commerce Lessons from Web Server Logs, 2nd ACM Conference on Electronic Commerce (EC'00), pp. 38-43, 2000.
[Rourke et al. 2001] Rourke L., Anderson T., Garrison D.R., Archer W., Methodological Issues in the Content Analysis of Computer Conference Transcripts, International Journal of Artificial Intelligence in Education, pp. 8-22, 2001.
[Rushton et al. 2006] Rushton J.P., Brainerd C.J., Pressley M., Behavioral Development and Construct Validity: The Principle of Aggregation, Psychological Bulletin, 94(1), pp. 18-38, 1983.
[Scambray et al. 2006] Scambray J., Shema M., Sima C., Hacking Exposed: Web Applications Second Edition, San Francisco, CA, USA, McGraw-Hill, 2006.
[Scott and Sharp 2002] Scott D., and Sharp R., Abstracting Application-level Web Security, In Proc. of the 11th Intl. Conference on the World Wide Web (WWW 2002), Honolulu, Hawaii, USA, pp. 396-407, 2002.
[Shankar et al. 2001] Shankar U., Talwar K., Foster J. S., Wagner D., Detecting format string vulnerabilities with type qualifiers, In 10th USENIX Security Symposium, Washington, D.C., pp. 201-220, 2001.
[Shapiro and Wilk 1972] Shapiro S.S. and Wilk M.B., An analysis of variance test for the exponential distribution, TechnoMeterics, Vol. 14, pp 355-370, 1972.
[Shiflett 2008] Shiflett C., Cross-site request forgeries, http://shiflett.org/articles/security-corner-dec2004, last accessed January 20, 2008.
[Shiflett 2004] Shiflett C., PHP Security, O'Reilly Open Source Convention, Portland, Oregon, USA, 26 Jul 2004.
[Simon 1955] Simon H.A., A Behavioral Model of Rational Choice, Quarterly Journal of Economics, 69(1), pp. 99-118, 1955.
[Spiliopoulou 2000] Spiliopoulou M., Web usage mining for Web site evaluation, Communications of the ACM, 43(8), pp. 127-134, 2000.
[Spiliopoulou et al. 2003] Spiliopoulou M., Mobasher B., Berendt B., Nakagawa M., A framework for the evaluation of session reconstruction heuristics in Web usage analysis. INFORMS Journal of Computing, 15(2), pp. 171-190, 2003.

[Spitzner 2001] Spitzner L., Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community, Chapter 6, Addison-Wesley, 2001.
[Squillante et al. 1999] Squillante M.S., Yao D.D., Li Z., Web traffic modeling and Web server performance analysis, Proceedings of the 38th IEEE Conference on Decision and Control, Vol.5, pp. 4432-4439, 1999.
[Stevens and D'Agostino 1986] Stevens M.A., D'Agostino R.B., Goodness of Fit Techniques, Marcel Dekker, New York 1986.
[Su and Wassermann 2006] Su Z., Wassermann G., The Essence of Command Injection Attacks in Web Applications, In The 33rd Annual Symposium on Principles of Programming Languages, Charleston, South Carolina, USA, pp. 372–382, 2006.
[Sumner et al. 2005] Sumner M., Frank E., Hall M., Speeding up Logistic Model Tree Induction, In: 9th European Conference on Principles and Practice of Knowledge Discovery in Databases, pp. 675-683, 2005.
[Sutton et al. 2007] Sutton M., Greene A., Amini P., Fuzzing: Brute Force Vulnerability Discovery, Addison-Wesley Professional, 2007.
[Swiderski and Snyder 2004] Swiderski F., Snyder W., Threat Modeling, Redmond, Washington, USA, Microsoft Press, 2004.
[Tappenden et al. 2005] Tappenden A.; Beatty P.; Miller J.; Geras A.; Smith M., Agile security testing of Web-based systems via HTTPUnit, Agile Conference, 2005. Proceedings, pp. 29-38, July 2005.
[Tappenden et al. 2006] Tappenden A.F., Huynh T., Miller J., Geras A., Smith M.R., Agile Development of Secure Web-Based Applications, International Journal of Information Technology and Web Engineering, 1(2), pp. 1–24, 2006.
[Thompson 2003] Thompson H.H., Why Security Testing Is Hard, IEEE Security & Privacy, 1(4), pp. 83–86, 2003.
[Tian et al. 2004] Tian J., Rudraraju S., Li Z., Evaluating Web Software Reliability Based on Workload and Failure Data Extracted from Server Logs, IEEE Transactions on Software Engineering, 30(11), pp.754-769, 2004.
[Tip 1995] Tip F., A survey of program slicing techniques. Journal of Programming Languages, 3(3), pp. 121-189, 1995.
[Trivedi 2001] Trivedi K.S., Probability and Statistics with Reliability, Queuing, and Computer Science Applications, second ed., John Wiley & Sons, 2001.
[Tsai et al. 2009] Tsai CF, Hsu YF, Lin CY, Lin WY, Intrusion detection by machine learning: A review, Expert Systems with Applications, 36(10), pp. 11994-12000, 2009.
[Tsourti and Panaretos 2001] Tsourti Z., Panaretos J., Extreme Value Index Estimators and Smoothing Alternatives: Review and Simulation Comparison, Athens University of Economics and Business, Statistics Technical Report No. 149, 2001.
[Valeur et al. 2005] Valeur F., Mutz D., Vigna G., A learning-based approach to the detection of SQL attacks, Intrusion and Malware Detection and Vulnerability Assessment, pp. 123-140, 2005.
[Vaswani 2000] Vaswani V., MySQL: The Complete Reference, McGraw-Hill/Osborne, 2004.
[Wagner et al. 2000] Wagner D., Foster J. S., Brewer E. A., and Aiken A., A First Step towards Automated Detection of Buffer Overrun Vulnerabilities, Network and Distributed System Security Symposium, San Diego, CA, pp. 3–17, 2000.
[Wang and Tang 2003] Wang W. and Tang M., User-Oriented Reliability Modeling for a Web System, 14th International Symposium on Software Reliability Engineering, pp. 293-304, 2003.
[Weiser 1984] Weiser, M., Program slicing, IEEE Transactions on Software Engineering, SE-10(4), pp. 352-357, 1984.
[Weiss and Provost 2003] Weiss G., Provost F., Learning when training data are costly: The effect of class distribution on tree induction, Journal of Artificial Intelligence Research, Vol. 19, pp. 315-354, 2003.
[Wheeler 2003] Wheeler, D., A., Secure Programming for Linux and Unix HOWTO, http://dwheeler.com/secure-programs, 2003.
[Whittaker and Thompson 2003] Whittaker J., Thompson H., How to Break Software Security, Addison-Wesley, 2003.

[Widenius and Axmark 2002] Widenius, M., Axmark, D., MySQL Reference Manual, Sebastopol, Calif.: O'Reilly, 2002.
[Wiegers 1999] Wiegers K., Software Requirements, Microsoft Press, Redmond, 1999.
[Witten and Frank 2005] Witten I.H., Frank E., Data Mining: Practical Machine Learning Tools and Techniques, Morgan Kaufmann, 2005.
[Williams and Lane 2002] Williams H.E., Lane D., Web Database Applications with PHP & MySql, O'Reilly, 2002.
[Williams 2001] Williams J., Avoiding the CNN Moment, IT Pro, March-April, pp. 68–72, 2001.
[Woodward et al. 1980] Woodward M. R., Hedley D., Hennel M.A., Experience with path analysis and testing of programs, IEEE Trans. Software Eng., SE-6(3), pp. 278-286, May 1980.
[Xie and Aiken 2006] Xie Y., Aiken A., Static Detection of Security Vulnerabilities in Scripting Languages, Proceedings of the 15th conference on USENIX Security Symposium, Article 13, 2006.
[Xu et al. 2005] Xu W., Bhatkar S., Sekar R., Practical dynamic taint analysis for countering input validation attacks on web applications., Technical Report SECLAB-05-04, Department of Computer Science, Stony Brook University, May 2005.
[Yeung and Ding 2003] Yeung DY, Ding Y, Host-based intrusion detection using dynamic and static behavioral models, Pattern Recognition, 36(1), pp. 229-243, 2003.
[Zhang et al. 2002] Zhang X., Edwards A., Jaeger T., Using CQual for static analysis of authorization hook placement. In the Proceedings of the 11th USENIX Security Symposium, San Francisco, CA, USA, pp. 33-48, 2002.
[Zipf 1949] Zipf G.K., Human Behavior and the principle of least effort, Addison-Wesley, 1949.

Appendix 1 – Introduction to Heavy-Tailed and Pareto Distributions

The majority of statistical work is based on short-tailed distributions such as the normal and lognormal distributions. These distributions decay "quickly" (commonly exponentially) in contrast with heavy-tailed distributions. The rank size law[53] (1949) can be used to informally describe heavy-tailed distributions. This law states that: the second largest entity is half the size of the largest; the third largest entity is one third the size of the largest, etc. That is, if the entities are ranked from largest (rank 1) to smallest (rank *n*), and their values are denoted as:

$$x_1 \geq \ldots \geq x_n$$

the rank *i* for an entity of value x_i is proportional to the proportion of entities greater than *i*. Or:

$$x_i \approx \frac{k}{i} \quad \text{(A1.1)}$$

for some constant *k*. More formally, Resnick (1997) states that a random variable X has a Pareto tail with index α, α > *0*, if for *x* > *1*

$$P[X > x] \approx x^{-\alpha} \qquad , x > 1 \quad \text{(A1.2)}$$

Many authors provide a slightly more generic distribution of a Pareto distribution by incorporating an additional multiplicative term x_{min}^{α} (the location parameter, the actual term is *L(x)*. For the Pareto distribution, *L(x)*= x_{min}^{α}), where x_{min}^{α} is a positive minimal value of *X; i.e.* $\forall x \bullet x > x_{min}^{\alpha}$.

Examination of the Pareto distribution (which is a commonly examined heavy-tailed distribution) involves analysis of the tail index α. Hence, α is examined with the common approach of setting $x_{min}^{\alpha} = 1$ and the requirement for the additional inequality (Equation A1.2). Technically, the above distribution is defined in a continuous domain; however, within this investigation's domain, the estimation of values clearly has a defined resolution. So strictly speaking X is a discrete random variable; and the discrete probability distribution analogue to the Pareto distribution applies. Therefore, the zeta distribution, or the Zipf distribution, is the actual distribution under analysis. However, the distributions only differ in their definition of the multiplicative term *L(x)* and hence the above definition resolves the issue of having a distribution defined in a continuous domain being applicable on a discrete random variable.

The Pareto distribution is an example of a wider set of distributions, namely heavy tailed distributions. X has a heavy tailed CDF *F(x)* if

[53] The rank size law is a good approximation for entities of high rank, but not for the largest.

$$1 - F(x) = P[X > x] = x^{-\alpha} L(x) \quad \text{(A1.3)}$$

where L is slowly varying; i.e.

$$\lim_{t \to \infty} \frac{L(tx)}{L(t)} = 1 \quad \text{(A1.4)}$$

The Pareto distribution is the "simplest" example of a heavy-tailed distribution and is used throughout this paper; and hence the more general definition can be considered solely for information purposes.

The implications of deciding that X is from a heavy-tailed or Pareto distribution are severe as the definition of the standardized moments become problematic. For the Pareto distribution, the first two moments are defined as:

$$E(X) = \frac{\alpha x_{\min}}{\alpha - 1} \quad \text{(A1.5)}, \qquad Var(X) = \frac{\alpha x_{\min}^2}{(\alpha - 2)(\alpha - 1)^2} \quad \text{(A1.6)}$$

This implies that, for $\alpha \leq 1$ the expected value is infinite; and for $\alpha \leq 2$ the variance infinite. Clearly, this demonstrates serious limitations on the types of models which can be constructed using Pareto distributed variables. In addition, these definitions are unrealistic in many situations because the distribution of X will be bounded by physical constraints. Hence, a more rigorous and realistic definition requires the above to hold over a finite range $[x_i, x_{i+j}]$ where the distribution applies.

Although this might seem an unimportant technical point, it is actually a recurring theme in this domain. Basically, all common methods of exploring potentially Pareto distributed variables follow this pattern where the investigation is only carried out within a finite range. Hence, the approaches introduce a bias because they only investigate a small component of the distribution, namely the "tail". x_i is often considered to be the start of the tail, although there is no method of evaluating *i* and no definition of the term *tail*. x_{i+j} is commonly considered to be near x_n; i.e. the highest ranked point within the data set. Clearly, the points, which in theory exist with ranks greater than *n*, cannot be inferred. It is important to note that this range only corresponds to an extremely finite part of the distribution; it is not uncommon for the "tail component" or Pareto range to be defined for less than 1% of the sampled range $x_1 \geq \ldots \geq x_n$. Hence, it is exceptionally difficult to make accurate estimations and infer reliable facts across such amounts of data. The amounts of data are very small both in absolute terms (the raw number of points) and relative terms (the percentage of the total sample). Hence, given the difficulty of accurately characterizing information as belonging to a heavy-tailed distribution and the significant consequences in terms of undefined standardized moments, one should be careful in inferring that a heavy-tailed distribution exists.

It should not be inferred from this discussion that the shape of the Pareto, or heavy-tailed, distributions are highly distinctive from short-tailed distributions. In fact, many heavy-tailed and short-tailed distributions "look" highly similar. For

example, Gong et al. (2001) plot the data from the Crovella and Bestavros (1997) paper, a time-series which contains file sizes transferred over a period of time. They compare the data at the 95% confidence intervals for both Pareto and lognormal models; and observe that the confidence intervals of both models grow with file size; and, at the tail, the two confidence intervals have a "large overlap which makes it difficult to distinguish them". Mathematically, Pareto and lognormal distributions also have a lot in common.

Adapting from Gabaix (1999) and Gong et al. (2001), consider a time series of i.i.d. positive random variables $Z_1, \ldots, Z_t, \ldots Z_\infty$. Let Z_i be defined as:

$$Z_t = Z_{t-1} A_t, \text{ t} = 1, \ldots. \quad \text{(A1.7)}$$

With $Z_0 = 1$. Taking logarithms yields

$$\ln Z_t = \sum_{i=1}^{t} A_i \text{ , t} = 1, \ldots.. \quad \text{(A1.8)}$$

which by the central limit theorem converges in distribution to a normally distributed random variable. Consequently, Z_t converges in distribution to a lognormal distributed random variable. Now, let's add a condition that Z_t must always exceed a threshold Δ.

$$Z_t = \max\{Z_{t-1} A_t, \Delta\}, t = 1, \ldots. \quad \text{(A1.8)}$$

Gabaix (1999) shows that Z_t now converges to a random variable with a Pareto distribution. That is, if $\Delta = 0$, it produces a lognormal distribution, otherwise a Pareto distribution. Because of the similarity between the two distributions, this paper also examines the lognormal distribution for the session lengths recorded.

This page is intentionally left blank.

Appendix 2 – Independence of Data Test for Chapter 7

Extreme value analysis methods are techniques that attempt to model rare events based on limited data. Heavy-tail analysis requires a dataset of unobtainable size; and hence, the analysis performed in this paper can be classified as extreme. Many extreme value analysis methods assume that the data set is independent. In fact, the Hill estimator is the only known estimator to perform accurately with dependent data (Rezaul and Grout 2006, Tsourti and Panaretos 2001). Hence, if the data is considered as dependent, extreme value analysis methods need to be modified. Therefore, in this appendix, this question is considered; however, in this situation, the definition and associated tests for independence is an extremely complex subject with no single clear answer. Independence or randomness is one of the four assumptions that typically underlie all measurement processes. The randomness assumption is critically important because most standard statistical tests depend on it; the validity of the test conclusions are directly linked to the validity of the randomness assumption.

To illustrate this issue, the autocorrelation function (*ACF*) is used to test for randomness or dependence of the data set. While an autocorrelation approach to the question is used, other approaches exist (see Brockwell and Davis (1991) for a discussion of alternatives). The session length data can be seen as a time-series because each session length is recorded according to the session start time. If the time-series is completely random then the entire *ACF* should be zero or the null hypothesis is *ACF(k) = 0*; where *k* is the lag. Examining *ACF* values, and determining if they are within the 95% confidence bounds around this central value is commonly utilized as a mechanism to test this hypothesis. If there are values exceeding this bound, then the data is considered dependent. Figures A2.1a and A2.1b show the *ACF* plots for ECE and Site A.

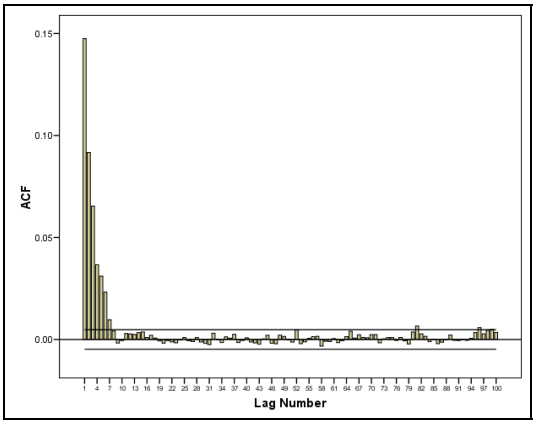


Figure A2.1a ACF for ECE

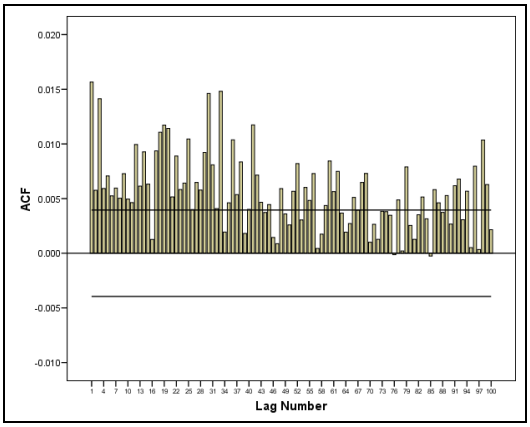


Figure A2.1b ACF for Site A

These plots also contain the 95% confidence bounds; the plots show that 10% and 67% of the values exceed the upper bound for the ECE and Site A sites

respectively, implying that the data may be dependent. However, the analysis uses Barlett's formula (Pankratz 1983) to estimate the confidence interval. This formula assumes that the data is normally distributed, and hence the confidence bounds are meaningless if the samples are drawn from a heavy-tailed distribution. Alternatively, the Ljung-Box test (Ljung and Box 1978) can be used to evaluate the null hypothesis. The Ljung-Box test utilizes the following formula:

$$Q = n(n+2)\sum_{k=1}^{m} \frac{acf_k}{n-k} \quad \text{(A2.1)}$$

where ACF_k is the ACF value for lag k, n is the number of samples and m is the maximum lag. Q is distributed as χ^2 with (m-p-q) degrees of freedom. The assumption that that $p = q = 0$ is made; i.e. that the data sets have no trend or periodic information. Clearly, this assumption is invalid as web-sites clearly have many different types of periods with differing resolutions; e.g. day/night; weekday/weekend; non-holiday-period/holiday-period etc. However, the exact nature of the periodic information is not understood and approaches to estimating p and q can be error prone. Hence this simplifying assumption is used. This assumption effectively inflates the Type II error; which is considered an acceptable risk in this situation. Using the above equation χ^2 is calculated to be 6582.68 and 586.88 for ECE and Site A respectively. These χ^2 values, with 100 degrees of freedom, correspond to a *p-value* of $p < 0.001$ for both websites. Hence, the null hypothesis can again be rejected which means that the data set is dependent, but only if it is not sampled from a heavy-tailed distribution. While this approach can be considered less distributionally restrictive than the previous approach, it is still, both theoretically (Jansen and de Vries 1991) and empirically (Chen 2002), not robust to heavy-tailed data.

In addition, the standard *ACF* formula is invalid if the sample is from a heavy-tailed distribution as the formula basically measures deviations from the sample mean, while the sample mean is mathematically undefined for many heavy-tailed distributions. Fortunately, the construction of a non-centered autocorrelation function is straightforward (Davis and Resnick 1985):

$$ACF_{HT}(k) = \frac{\sum_{i=1}^{n-k} X_i X_{i+k}}{\sum_{i=1}^{n} X_i^2} \quad \text{(A2.2)}$$

Figures A2.2a and A2.2b shows the *heavy-tailed ACF* plots for ECE and Site A. These plots show that the *ACF* values do not exceed 0.17 and 0.13 for the ECE and Site A sites respectively. However, confidence bounds estimations (or Q statistics) no longer exist; and unless specific information about the underlying distribution, including accurate values for its parameters, are known, a confidence interval cannot be defined (Feigen and Resnick 1999).

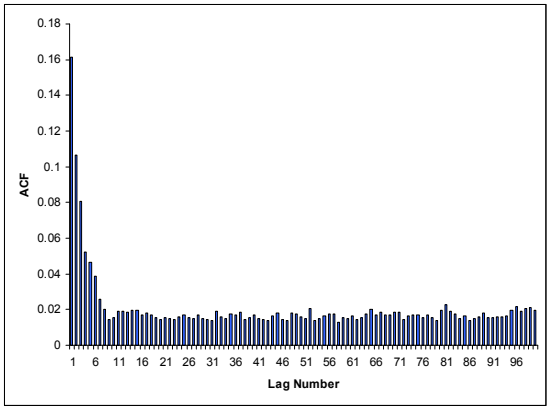


Figure A2.2a Heavy-Tailed ACF for ECE

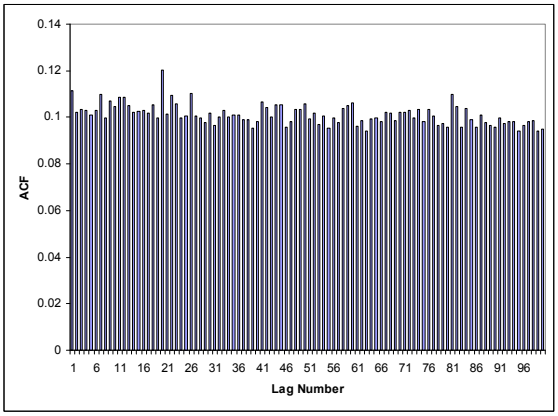


Figure A2.2b Heavy-Tailed ACF for Site A

However, several alternative approaches still exist for evaluating the null hypothesis. Feigin and Resnick (1999) show that if the series can be modeled as a moving average process of lag l then the coefficients of the *heavy-tailed ACF* should decay to approximately zero beyond l; and in the limiting case where $l \to \infty$, the coefficients should again all be approximately zero. This question can be investigated by asking if the co-efficients are summable. In addition, a more formal test can be constructed by forming a permutation distribution. *The heavy-tailed ACF's* behavior, with respect to the null hypothesis, can be characterized by a summary statistic; e.g. the *maximum absolute ACF coefficient*[54]; this option is recommended by Feigen and Resnick (1999). The *p-value* of the observed summarizing statistic is estimated by generating 999 permutations of the time-series; computing the statistic for each permutation and counting the number (C) of values greater than or equal to the actual observed statistics. The *p-value* is given approximately by *((1+C)/1000)*. Clearly, this approach avoids relying in the asymptotic theory or distribution for this particular summarizing statistic; and the test is distributionally robust for heavy-tailed situations. Figures A2.3 and A2.4 display the results of the permutation test.

[54] Other options include the partial or biserial autocorrelations.

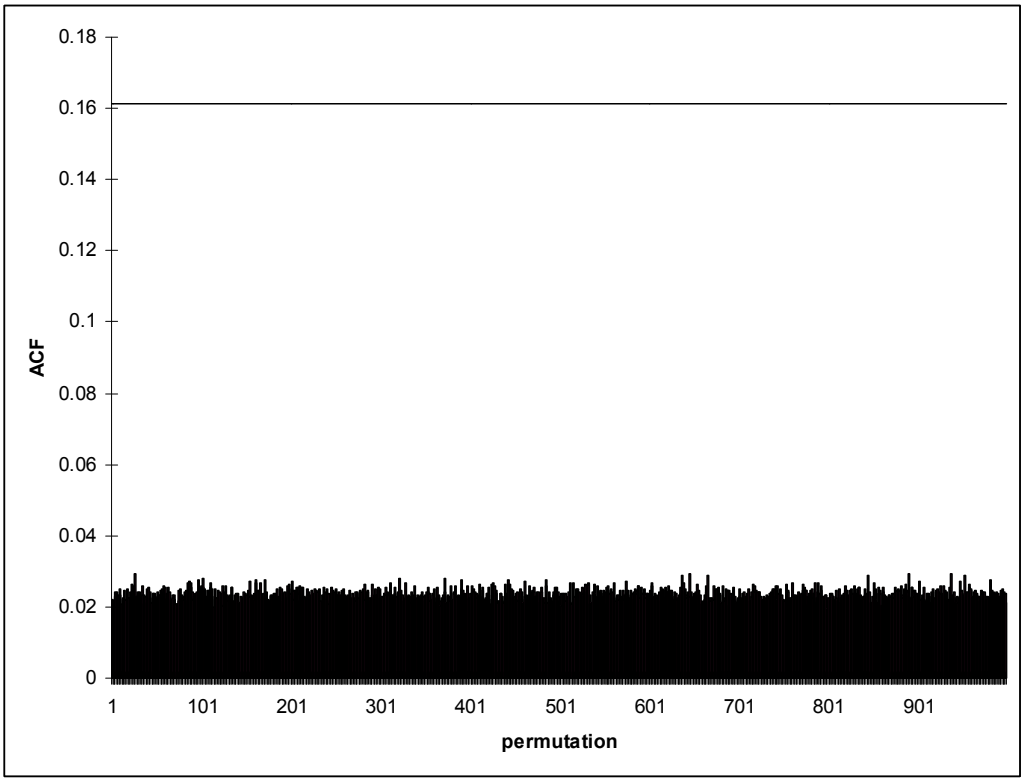


Figure A2.3 Permutation test for ECE

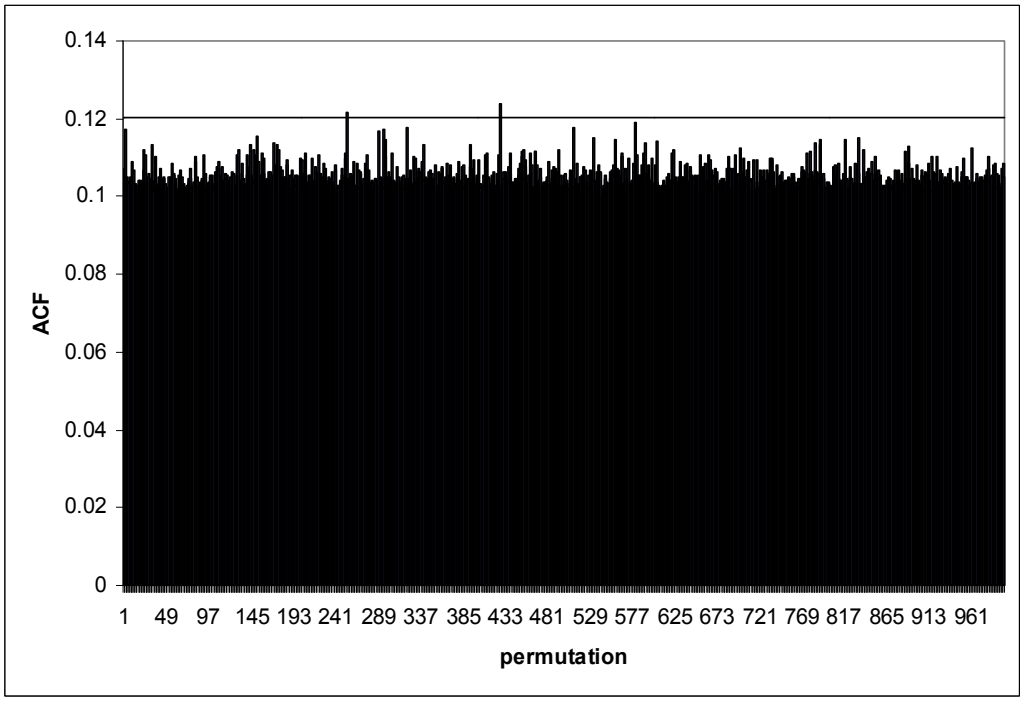


Figure A2.4 Permutation test for Site A

Visual inspection show that the majority of the *max(ACF)* of the permutations are below the actual *max(ACF)* which is represented by the horizontal line. In fact, for the ECE site, none of the permutations are greater than or equal to the actual *max(ACF)* which means the *p-value* < 0.001. For Site A, two of the permutations are greater than or equal to the actual *max(ACF)*; hence, the *p-value* < 0.003. Because the *p-value* for both websites are below the standard type I error cut-off values, the null hypothesis can be rejected which means that the data for both websites are dependent.

While this approach is now a relatively robust examination of the null hypotheses several situations still exist where the validity of the approach and hence the associated results are at best questionable and at worst non-applicable. Feigin and Resnick (1999) empirically demonstrate that the *heavy-tailed ACF* tends to exhibit erratic results in the following situations:

- the presence of any non-linearities, such as the process being a bilinear process;
- when the process is a moving average*(l)* process; if $l > m$;
- the series is contaminated by (additive) outliers.

These situations clearly represent risks to the internal validity of the results presented in this appendix.