A FORENSIC WEB LOG ANALYSIS TOOL: TECHNIQUES AND IMPLEMENTATION

Ann Fry

A thesis
in
The Department
of
Concordia Institute for Information Systems Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Information Systems Security
Concordia University
Montréal, Québec, Canada

September 2011

Concordia University

School of Graduate Studies

This is to certify that the thesis prepared

By:	**Ann Fry**
Entitled:	**A Forensic Web Log Analysis Tool: Techniques and Implementation**

and submitted in partial fulfillment of the requirements for the degree of

Master of Information Systems Security

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

____ Chair
Dr. Benjamin Fung

____ External Examiner
Dr. Otmane Ait-Mohamed

____ Examiner
Dr. Amir Youssef

____ Supervisor
Dr. Mourad Debbabi

Approved ____________________Dr. Mourad Debbabi ____________________
Chair of Department or Graduate Program Director

____________ 20 _____ ____

Dr. Nabil Esmail, Dean

Faculty of Engineering and Computer Science

Abstract

A Forensic Web Log Analysis Tool: Techniques and Implementation

Ann Fry

Methodologies presently in use to perform forensic analysis of web applications are decidedly lacking. Although the number of log analysis tools available is exceedingly large, most only employ simple statistical analysis or rudimentary search capabilities. More precisely these tools were not designed to be forensically capable. The threat of online assault, the ever growing reliance on the performance of necessary services conducted online, and the lack of efficient forensic methods in this area provide a background outlining the need for such a tool. The culmination of study emanating from this thesis not only presents a forensic log analysis framework, but also outlines an innovative methodology of analyzing log files based on a concept that uses regular expressions, and a variety of solutions to problems associated with existing tools. The implementation is designed to detect critical web application security flaws gleaned from event data contained within the access log files of the underlying Apache Web Service (AWS).

Of utmost importance to a forensic investigator or incident responder is the generation of an event timeline preceeding the incident under investigation. Regular expressions power the search capability of our framework by enabling the detection of a variety of injection-based attacks that represent significant timeline interactions. The knowledge of the underlying event structure of each `access_log` entry is essential to efficiently parse log files and determine timeline interactions. Another feature added to our tool includes the ability to modify, remove, or add regular expressions. This feature addresses the need for investigators to adapt the environment to include investigation specific queries along with suggested default signatures. The regular expressions are signature definitions used to detect attacks toward both applications whose functionality requires a web service and the service itself. The tool provides a variety of default vulnerability signatures to scan for and outputs resulting detections.

Acknowledgments

I would like to thank Dr. Mourad Debbabi, my advisor, for his continuous support and invaluable guidance throughout my academic program. He welcomed me to the Forensics research team in September of 2006. He gave me the freedom to pursue independent ideas, while challenging me with “why, what, how” from time to time to help me clarify my mind. My appreciation extends to the members on my graduation committee: Dr. Amir Youssef, Dr. Otmane Ait-Mohamed, and Dr. Benjamin Fung. Their valuable suggestions helped enhance the content of this thesis. My sincere thanks to my fellow researchers: Adam, Marc-Andre, Nadia, Hatim (R.I.P.), Asaad and Ali. I would also like to thank in particular Lehan Meng for assistance completing the finishing touches on the implementation, reference checking and an effective user guide. Special appreciation is extended to Yosr Jarraya for assistance in corrections and revisions. Special thanks go to Emerson, Simon, Ali, Simsong, Metr0, gdead, Moose and Daemons. Finally and most importantly, my deepest appreciation goes to my parents and my family, without whose support this would not have been possible. Mom, Dad, Tricia, Mike, Joey, Liam and Elliot; the furry members, Luna, Sunshine, Murphy, Merlin, Nexus (R.I.P.) and Orion (R.I.P.); I love you. Many thanks to them for their continuous support, understanding, licks and nudges of encouragement insisting that I get back to work.

Contents

List of Figures

List of Tables

Glossary

Cross Site Request Forgery A good description of this attack is provided within Section 4.4. xvi

Distributed Denial Of Service A Denial Of Service (DoS) attack that is initiated and executed from multiple origin points. xvii

Denial Of Service Any action which causes service disruption, or a user's inability to access the service of an intended target comprises a denial of service attack. Excess traffic on the network or directed to the service machine, a shutdown of the machine or services that the target application relies on, or removal or modification of data or code comprising dependancies for a target application are all examples of how to cause a denial of service attack. xvii

Cross Site Scripting A good description of this attack is provided within Section 4.5. xxi

Brute Force Brute force is a paradigm where all possible cases for deriving the solution to a problem is explored using an exhaustive search [392]. This method does not employ any complex heuristics to alter the search space, and instead relies entirely on computational power. 17

Keystroke Logger An implementation that performs keystroke logging may be software or hardware based. It provides a record of key press events within a log file on a target machine and usually operates covertly. 11

LAMP A LAMP web application base consists of the Linux[279] operating system, an AWS, the script language PHP:Hypertext Preprocessor (PHP)[194] and a MySQL[101] database. 36

Log Rotation Effective log management begins with the ability to manipulate log files [2]. Log file manipulation becomes easier when the size of the log file remains limited. Log rotation ensures log files do not grow past a certain size through the creation of an additional file once the log reaches a specified file size. 10, 20, 21

Phishing attacks involve manufacturing a clone of a target application or website and persuading unsuspecting users to interact with the malicious site[360]. 35, 38

Spam A good description of this attack is provided within Section 4.7. 21, 60

Spider or Spidering A spider works by traversing a web site, one page at a time, gathering, storing and reporting relevant meta-data specified by its operator. Information relevant to the operation of spiders include email addresses, links, form data, tag data, among others [59]. 14, 32

Acronyms

`CR` Carriage Return. 40, 41

`LF` Line Feed. 40, 110, 111

ADO Active Directory Object. 59

AIDE Advanced Intrusion Detection Environment. 64

AIX Advanced Interactive eXecutive. 75

ANSI American National Standards Institute. 39, 40

API Application Programming Interface. 68, 69

ARPANET Advanced Research Projects Agency Network. 108

ASCII American Standard Code for Information Interchange. 39–41, 69, 90

ASP Active Server Pages. 58, 105

AWS Apache Web Service. iii, xiv, 6, 11, 13, 15–22, 25–30, 37, 52, 53, 60, 63, 64, 68, 83–85, 113, 115, 116, 122, 125, 126

BCC Blind Carbon Copy. 110, 111

CC Carbon Copy. 110, 111

CGI Common Gateway Interface. 18, 19, 21, 29, 54

CLF Common Log Format. 21, 26, 27, 29, 53, 58, 61, 67, 83

CSRF Cross Site Request Forgery. xiv, 85, 96, 97, 99

CSS	Cascading Style Sheets. 13, 146
CSV	Comma Separated Values. 56, 59, 65, 70, 82
DDE	Dynamic Data Exchange. 68
DDL	Database Definition File. 68
DDoS	Distributed Denial Of Service. xiv, 17, 33
DEC	Digital Equipment Corporation. 108
DHCP	Dynamic Host Configuration Protocol. 56
DLF	Distilled Log Format. 58, 60
DNS	Domain Name Service. 59–61
DOM	Document Object Model. 13, 102
DOS	Disk Operating System. 40
DoS	Denial Of Service. xiv, 17, 64, 84, 106
EJBQL	Enterprise JavaBeans Query Language. 70
ELF	Extended Logfile Format. 26, 53, 58, 67, 75, 83
ESM	Enterprise Security Management. 72, 81
ETL	Extraction, Transformation and Loading. 54
FISMA	Federal Information Security Management Act. 57
FTK	Forensic ToolKit. 82, 124
FTP	File Transfer Protocol. 53, 56, 57, 60, 61, 75
Gb	Gigabyte. 122
GLBA	Gramm-Leach-Bliley Act. 57
GUI	Graphical User Interface. 62, 75, 82, 113, 114

HIMS Host Integrity Monitoring System. 3, 73–75

HIPAA Health Insurance Portability and Accountability Act. 57

HP Hewlett Packard. 38

HQL Hibernate Query Language. 70

HTML Hyper-Text Markup Language. 13, 23, 32, 51, 52, 54–58, 60, 61, 66, 70, 82, 98, 99, 103, 105, 111, 147

HTTP Hyper Text Transfer Protocol. 10, 11, 13, 14, 17, 18, 22, 25–32, 34, 37, 56, 57, 59, 60, 67, 75, 79, 85, 92, 98, 121

HTTPS Hyper Text Transfer Protocol Secure. 17

IBM International Business Machines. 56

IDS Intrusion Detection System. 3, 5, 6, 10, 68, 71–76

IE Internet Explorer. 11–13, 96, 105

IETF Internet Engineering Task Force. 12

IIS Internet Information Services. 11, 52, 53, 56, 59, 75, 82, 90

IP Internet Protocol. 12, 13, 21, 27, 31, 34, 65, 67, 109

IPS Intrusion Protection System. 72, 75, 76

IPV4 Internet Protocol Version 4. 61

IPV6 Internet Protocol Version 6. 61

ISAPI Internet Server Application Programming Interface. 4

ISP Internet Service Provider. 18, 30

J2EE Java 2 Enterprise Edition. 69, 70

JDO Java Data Object. 68

JS Javascript. 11, 12, 14, 36, 61, 99, 102, 144, 146, 147

JSON JavaScript Object Notation. 97

kb kilobyte. 122

LDAP Light-weight Directory Access Protocol. 85–87

MDAC Microsoft Data Access Components. 58

MDX Multi-Dimensional eXpressions. 70

MFSA Mozilla Foundation Security Advisory. 99

MS Microsoft. 9, 11–13, 35, 38, 53, 56–60, 75, 134, 138, 139, 141

ms milliseconds. 122

MTA Message Transfer Agent. 52, 53

NCSA National Center for Supercomputing Applications. 26, 59, 61, 75

ODBC Open DataBase Connectivity. 135

ODF OpenOffice Data Format. 70

OJB Object Relational Bridge. 68

OLAP OnLine Analytical Processing. 54, 69

OPC Object Linking and Embedding for Process Control. 68, 69

OWASP Open Web Application Security Project. 1, 96

PCI Payment Card Industry. 75

PCI-DSS Payment Card Industry - Data Security Standards. 57

PDA Personal Digital Assistant. 14

PDF Portable Document Format. 54, 56, 58, 70, 82

PGP Pretty Good Privacy. 74

PHP PHP:Hypertext Preprocessor. xiv, 61, 67, 68, 93, 107

PHPIDS PHP:Hypertext Preprocessor Intrusion Detection System. 84

pid process ID. 28

PIPEDA Personal Information Protection and Electronic Documents Act. 57

PL/SQL Procedural Language / Structured Query Language. 70

REST Representational State Transfer. 36, 61

RSS Really Simple Syndication. 62

RTF Rich Text Format. 58, 70, 82

SEC Simple Event Correlator. 70, 71

SEM Security Event Management. 51

SIEM Security Information Event Management. 51

SIM Security Information Management. 51

SMS Short Message Service. 33

SMTP Simple Mail Transfer Protocol. 75, 109

SOX Sarbanes-Oxley Act. 57

SQL Structured Query Language. 56, 58–60, 63, 70, 81, 82, 85, 87, 89, 106, 107, 113, 115, 116, 118, 128, 132–134, 137–140

SSH Secure SHell. 60, 66

SSL Secure Sockets Layer. 29, 35, 73

SUID Set User IDentification. 74

TCP Transmission Control Protocol. 13, 21, 28, 31, 55, 68

tid thread ID. 28

TSV Tab Separated Values. 59, 82

UDP User Datagram Protocol. 31, 55

URI Uniform Resource Indentifier. 25, 99

URL Uniform Resource Locator. 14, 15, 22, 23, 26, 29, 30, 63, 64, 66, 85, 87, 90, 92, 93, 95–97, 102, 105, 125, 147, 148

VBScript Visual Basic Scripting edition. 99, 105, 106

VM Virtual Machine. 3

VPN Virtual Private Network. 3

W3C World Wide Web Consortium. 21, 52, 53, 56–59, 61, 75, 82, 102

XHTML eXtensible Hyper-Text Markup Language. 13, 54, 98

XML eXtensible Markup Language. 13, 59, 60, 68, 70, 82, 99, 102, 146

XSS Cross Site Scripting. xiv, 14, 36, 38, 53, 85, 99, 100, 103, 106, 147

Chapter 1

Introduction

The Internet, home to almost two billion users and over 156 million websites [8], has evolved from the simplest text and hyperlink based pages to today's standards of dynamic mobile productivity applications [191]. Due to increased demand of the means to access increasingly sophisticated applications over the Internet, the number of web applications developed recently has exploded for use across multiple platforms. Interest in web application security has risen dramatically relative to the number of vulnerable applications. There exists a charitable organization called the Open Web Application Security Project (OWASP) [59] that is dedicated to improving the security of web application related software. Web application history evolves from the use of simple scripts to richly designed user interfaces with both client and server side behaviors, written in increasingly complex languages. The evolution timeline of web application development is well illustrated in Figure 1 [247]

1.1 Motivations

With the evolution of web application technologies arises a revolutionary profile of attack trends. The current profile highlights the main issues affecting the state of website security for any potential attack target. The variety of dependencies upon which web applications rely as well as any correspondance between them comprise the security landscape of a target. These dependencies include the operating system foundation, back-end services such as databases and the web service itself, and potentially many other interoperable web applications. It has been determined that attackers

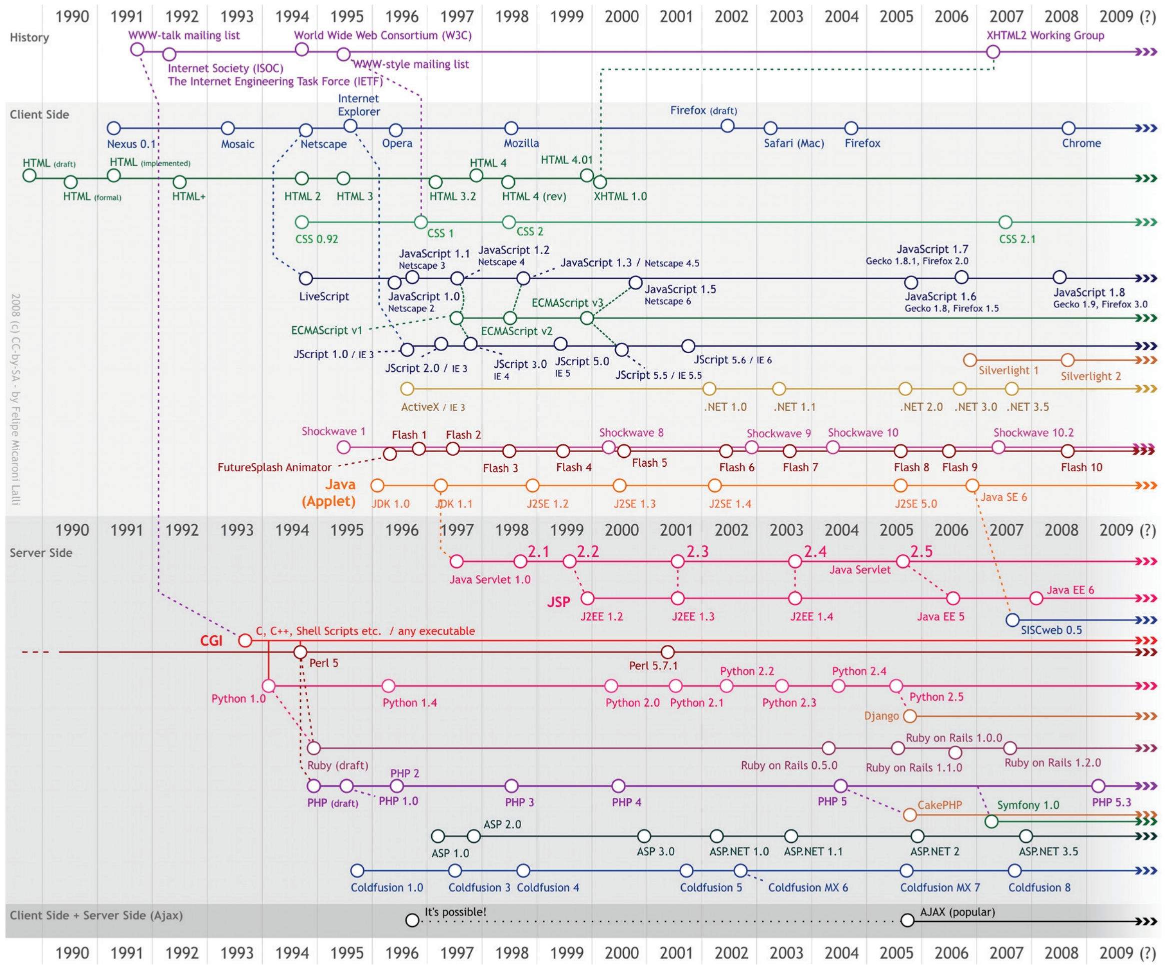


Figure 1: Web Development Timeline [247]

motivations are generally consistent [384]. However, one reason their exploit methods remain unpredictable is due to the large size of the security landscape. It is this same landscape that causes the tracking of these deviants to continue to be an ever increasingly complex task.

The ability to track the evolution of malicious parties through the association of individual attacks against a target system are essential to a incident response record of events. This record establishes that log analysis is an important and essential component to the investigation of an organization's security state from both administrative as well as forensic standpoints. Each network device such as a workstation, server, router, switch, Virtual Private Network (VPN), Virtual Machine (VM), firewall, Intrusion Detection System (IDS), or Host Integrity Monitoring System (HIMS) generates logs that contain records of system, device, and user activities that have taken place within the infrastructure. As such, analysis can provide explanations and determine trends for events such as login success or failure, website visits, files read, modified or deleted, amount of network bandwidth used, or identify imminent threats such as attacks, viruses or other network anomalies [262]. During normal operations, web applications may include a logging function to record event information. These are used to provide proof of validity that can be required by law or corporate policies. They are also used to ensure individual accountability in the application sphere by tracking a user's actions [58]. Even if a web application does not provide this functionality, most underlying service applications do provide very comprehensive logging capabilities.

Digital forensic science concerns itself with the collection, preservation and documentation of evidentiary data. It is the analysis of all data present on a computer system which aided the commission of a criminal offence. The data's ability to persuade with a high level of confidence that a particular action has occurred and its suitability of admission classifies it as forensic evidence [43]. The collection of such data after an incident targets a web application requires knowledge of the many interworking technologies associated with the application. Pairing the extensive logging capabilities of the web service application with the analysis of the underlying server enables the forensic analyst to get an improved survey of any incident. It is easily apparent for forensic purposes, that the analysis of these logged events is a crucial step in the investigation process. Existing log analysis solutions do not provide forensic functionalities necessary to analyze and classify data as evidence. These shortcomings comprise the basis of the problem we propose to solve, and are described below.

1.2 Problem Statement

Presently, with the increasing prevelance of crimes associated with digital devices, applying scientific processes to obtain evidentiary data has become a necessity. During an investigation where the prime evidence is contained within web service log events, one of the first problems faced by a digital investigator is the decision of how to organize and analyse log data of a target machine. The problem with current log analysis technologies is that they do not apply principles necessary to qualify as forensic analysis solutions. Forensic principles such as efficiency, processing large file sizes, classification of data, comprehensive reporting, modularity, extensibility and the detection of critical timeline events associated with an incident must be considered in the development of an appropriate solution.

To effectively perform web application forensics, it is necessary to ensure the validity of events and provide an agile method of analysis. With regards to efficiency, a size-based log rotation implementation such as `logrotate` [371] ensures log files that normally aggregate quickly are individually organized to be accessed efficiently [2]. Some web applications include the necessary components to provide the capability of log rotation such as Trac [352] or Twiki [309], while others do not. This is especially important in a live investigation, as this requires an application which is both time sensitive and capable of providing critical information. Several features of web applications ensure that an investigator cannot analyze the web application logs independantly from other sources to acquire all necessary information pertinent to an investigation.

One such feature is the fact that each web application may or may not come with logging capabilities. In the circumstance where an application does not generate events, errors are sent to the underlying server application log or another proprietary debug tool. Therefore, to aid in the validity or admissibility of events, a thorough understanding of the logging capabilities of the server itself should be attained. The logs provide a record of events occuring prior to or within the duration of an incident. Another feature is consumed with the idea that some web applications are built to perform logging such as the Internet Server Application Programming Interface (ISAPI) [157]; the verbosity and frequency of the events that are logged are application specific. All application log files are cryptic due to the nature of their inception as an invention of necessity to programmers during development. They were not designed to support incident investigations. Appropriate log management practices limit the challenges faced by forensic analysts.

With regards to the problem of web application forensics, most research and development has been performed by IDSs. Although intrusion detection research is applicable, various methodologies suggested by said research have yet to be directly applied to the area of web application related forensic analysis. In the following section are highlighted objectives that purport to solve the inadequacies of present log analysis solutions through the research and development of a tool incorporating forensic principles.

1.3 Objectives

The following outline of proposed objectives for this research incorporate the development of forensic principles and their application to a proposed forensic log analysis solution:

- Prepare an effective account of the engineering process that develops a forensic log analysis tool by studying the state of the art methods presently applied to log analysis by today's web application security specialists.
- Determine those principles necessary to the production of an implementation functionally appropriate for forensic log analysis of web applications. This includes details of the implementation, design and a description of applicable technologies.
- Elaborate a new technique for the analysis of web server logs for forensic investigation.
- Prototype the proposed technique for the purpose of validation.

1.4 Contributions

An outline of the contributions made by our study is included within the following section:

1. Provided as a solution to the lack of forensic log analysis tools, the implementation developed in conjunction with the preparation of this document implements functionalities according to several principles requisite to a web log file analyzer. The forensic principles left unaccomplished by current log analysis solutions include ensuring enough evidentiary material is present, fast processing time, the ability to process large file sizes, the organization of events, readable reports, a modular and extensible code base for reuse and adaptability and the association of detected critical events with the incident timeline. Ensuring evidentiary material is

already present on a target machine can prove difficult. To combat this principle our solution analyzes a resource already available to the investigator; the logging service of the underlying server application.

2. Our innovative approach that includes the use regular expressions for the performance of forensic web log analysis is the first of it's kind. Our methodology, one currently employed by IDSs; uses regular expressions to perform forensic analysis of AWS [164, 163] `access_log`s with regards to web applications. The use of regular expressions not only expediates the processing time of logs significantly, but also provides a more comprehensive search method than that normally found within log analysis solutions. Due to the efficiency of regular expressions, processing of files that contain copious amounts of data can be accomplished with minimal computational cost. Usage of a database in conjuction with the parsing capabilities of our tool enables a more organized and comprehensive retrieval of data for the purposes of reporting and analysis. Albiet such reporting is not implemented by our tool. The design of our implementation includes a modular aspect due to the requirement that it remain a plugin to the forensics framework. Code reuse may enable others to implement those features applicable in a forensic scenario undiscovered presently. The extensible nature of our tool includes he ability to add, modify or exclude particular regular expressions. This in turn not only allows for adaptability but also the application of fine grain control for particular investigation scenarios.

3. Due to the extensive comparative study of different log analysis solutions, insights into how successful exploits affect the timeline preceeding and during an incident are discovered. Submission of strings that can prove exploitational serve as distinctive markers when observing events in a timeline. The unauthorized execution of code segments can alter the appropriate functionality of a target machine in such a way as to compromise the system's integrity. It is this full compromise that comprises an incident. At which point a forensic investigation begins. These significant events have consequences, and impact the evolutionary timeline by altering states of the server. It is important within an investigation to know the states prior to, during and after the compromise of a system has taken place. For not only does this determine how much damage is done in a particular incident, but offers sources as to the cause, and with further exploration, can lead an investigator to the perpetrator of the devious acts. In conclusion, this variation of log file analysis has many limitations. However, it is believed to be a good starting point to develop future applications, that may involve a more rigorous

analysis methodology.

1.5 Organization

Within the second chapter, we provide an overview of the theories and technologies associated with and affected by the web vulnerabilities targeted in this study. The third chapter expresses an outline of related work studies that had an impact on our dissertation. In the fourth chapter we discuss the regular expressions used to describe the attack vectors detectable by our implementation. The forensic log analysis tool description devoted to chapter five outlines the design and implementation of a software environment that prototypes the proposed approach. Finally, some concluding results obtained during the course of this research with a discussion of future work are ultimately presented in chapter six.

Chapter 2

Preliminaries

As a topic applicable to modern Internet culture, we assume our reader is familiar with at least one web application. The prolific use of social networking sites, familiarity and ease of use to access online banking or to go shopping online all expose the general public to the face of web applications in everyday Internet usage. However what are web applications? How have they evolved? Is their expansion too rapid to ensure the security of our private information? Upon which technologies have they been developed? How can we view web application transactions forensically? These are all valid concerns. This brief overview of theories and technologies associated with the dissertation's subject matter, aims to provide the reader with the context necessary to answer these questions. With respect to each of the individual areas touched by the subject matter, this chapter looks to provide the reader with sufficient understanding of the methodology chosen for our proposed contributions. The areas discussed are cyber-forensics, web applications and their components, as well as regular expressions. The complexity of web application forensics increases with the number and variety of underlying components a single web application depends on. These components include network structure, operating system files structure as well as additional service dependencies. One common example of a service a web application would rely on is a database service. The interactions between and where vulnerabilities affect web application components are demonstrated in Figure 2.

2.1 Log Analysis

The evolution of digital evidence extraction from electronic devices continues to increase in complexity with the introduction of new products to consumers. With the ever increasing capacities

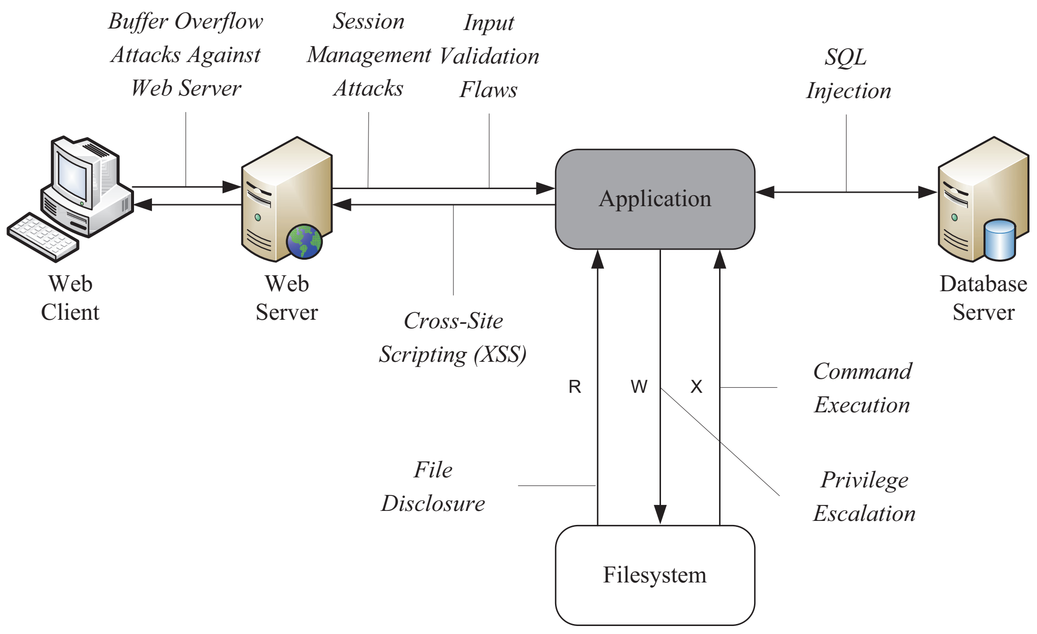


Figure 2: Web Application Architecture [329]

of storage devices, partitioning certain areas of memory for a more rigorous examination enables forensic tools to produce more relevant data to the investigation at hand. This selection process reduces the amount of data to be analyzed by removing redundancy, and illustrates the importance of having different methods of forensic inquiry for each type of data that requires analysis [333]. Due to developmental processes, most if not all software applications, operating systems, services and drivers have methods, which enable them to record or log information about events that occur during normal operation. Aggregations of event data are incredibly important to a forensic investigator during the time line reconstruction of an incident. Analysis of the information contained within log files can be used to accurately reconstruct user activities, and the methodology of accomplishing this has been performed on a variety of operating systems including Microsoft (MS) [81] based environments [349].

Although log files were not built specifically for forensic purposes, they are the most likely of all files resident on a system to contain the majority of evidentiary information. If they do not contain such information, they may provide links to other sources of information for an investigator to generate an accurate representation of the virtual environment at the time of the incident, also known as the "scene of the crime". Logs were generated by developers to report on services or applications during run time. Events contained within these logs can indicate both normal or erroneous operation of an application. The purpose of these events are to inform application developers of their presence. The lack of a particular notification could also indicate the occurrence of an error. The results of this application-specific log file include non-standard data types, event formats, rudimentary event category definitions, and file formats whose integrity cannot be verified. Other problems currently faced by log analysts include how to manage and analyze large files, and how to organize and report their findings in a format acceptable and understandable to a court of law. Currently, the best method of practice amongst log analysis professionals or system administrators is their own empirical research which uncovers event anomalies amid the vast collection of normally logged events. Log file analysis is a mandatory process that should be performed during all computer forensic investigations, however, due to the nature and potential size of various log files, this task is one of the most processing intensive.

Using default settings, a machine will not log all of the information that is beneficial to the forensic investigator. The machine setup and audit policies for all machines on the network must be fine tuned to the appropriate level for each application, service, and operating system task. This ensures all potentially admissible events are recorded. In an ideal forensic situation, an excellent system administrator of the network under investigation, would already have three systems in place to accomplish the following tasks [2]. The first machine would comprise a central stealth repository for all events to be gathered from across the network. Organizing all of the network events in one place enables efficient log rotation, backup capabilities, and easier log file to log file correlation. Ensuring that the central repository is setup in a stealth manner provides another level of security against log file tampering. The second machine would support the processing of an IDS. The third machine would perform any preliminary investigation and rudimentary analysis defined by the system administrator that would be beneficial for understanding the state of their network.

2.2 Web Applications

In this section, we aim to provide an overview of web applications and their security from a forensic perspective. The basic protocol Hyper Text Transfer Protocol (HTTP) [131] is stateless. Problems with web applications are partially due to the instantiation of increasingly complex protocols upon this unstable basis [329]. When designing secure web applications, developers must consider the inherent insecurity of HTTP. Unfortunately, this is commonly overlooked as developers tend to be overly concerned with deadlines rather than security. The categories of web applications, known to comprise the HTTP architecture, include the HTTP server, proxy, cache, gateway, tunnel, agent, and robot applications [109]. Applications which extend the functionality of these fundamental building blocks are also of concern to web application security professionals. In July of 2004, attackers staged several Internet Explorer (IE) [79] attacks designed to steal banking and credit card information. This occurred when malicous parties took over several MS Internet Information Services (IIS) web servers [88] worldwide, where flaws within IE composed of malicious Javascript (JS) [280] were automatically downloaded whenever a user visited an infected site. Once run, the JS downloaded a keystroke logger [347] trojan horse [289] from another server in Russia [377]. This attack relied on leveraging vulnerabilities in multiple web applications in order to complete the attack successfully. Both the client side browser and the service itself contained vulnerabilities. This shows that an attack does not just happen against the web application itself, but also on processes and technologies that formulate the foundation of web applications. Web browsers programmatically function as agents, caches, gateways and robot applications. They also provide the necessary configuration to enable the use of tunnels and proxies. A web server's primary function is to act as an HTTP server. Depending upon its configuration, a web server instance can also perform any of the additionally described HTTP architecture web applications. One such configuration could be used to harden a network. This would be achieved by implementing a gateway device that isolates the servers from the web and accepts the traffic on their behalf [329]. As an example of web service flexibility, the AWS can implement this functionality by operating as a reverse proxy. Subsequently, we will discuss server side applications and how vulnerabilities generated client side can be executed server side. As the number of web applications continues to grow, we will take a look at some of the more commonly used web applications that were considered within the scope of our research.

2.2.1 Client Side

The browser itself is a key component of the Internet because it represents the visual interface between the user and the Internet [110]. It enables the user to generate standard HTTP request messages in addition to the storage and tracking of session data. This session data is then accesssible to the browser, individual web resources or applications. The browser enables access of data resident on the platform itself. The browser's functionalities are not limited to traversal direction and data submission; distributed web applications employ the browser's ability to process data [360]. The browser's main purpose is to act as a gateway to the information on the Internet. It can also provide caching and configuration for using proxies. Each browser has a unique method of rendering the data it requests. The significant variations in the rendering process require web developers to process and test the behavior of their applications across multiple versions of the same product. Even slight version variation can cause undesirable functionality or security vulnerabilities. Web browsers vulnerabilities are also caused by the browsers functionalities necessary to interact with varying Internet content types and requisite processing capabilities. Although according to the graph at [22] where browser manufacturers claim their products to be secure against all of these types of attacks, the server providing the web service itself cannot fully be secured. Attacks may be made via different avenues to the server providing the web service. Even now vulnerabilities within browsers are still being reported to security mailing lists [278]. Other indications of these vulnerabilities arise when browser development teams release bug fixes or security updates to their products. For example, at the time of writing this MS just released the largest batch of updates since the inception of the first "Patch Tuesday" back in 2003. These updates are not only applicable to IE, but also to the MS Windows 7 [90] operating system and the MS Office suite [257]. Concerning these types of vulnerabilities, the developers are taking proprietary measures, not standardized with a governing body such as the Internet Engineering Task Force (IETF) [137] that do not completely mitigate all web related vulnerabilities. One problem with the acceptance of such a standardization is due to the nature of most Internet advertisements whose revenue generation is based on the number of users that have viewed a particular ad [46]. As stated, it is this variety of code that enables website owners to collect revenue for advertisements, or to collect information such as users' Internet Protocol (IP) [315] addresses for statisical purposes. The ability to collect this data cannot only be performed server side. The required functionality of the browser to execute code also opens up the possibility of malicious code being written into websites to be executed by the browser of an unsuspecting user. Providing users with security information when Java [186] is run from the browser raises awareness to the end users in the form of a warning. Unfortunately the browser's ability to run code such as JS or Adobe [68] Actionscript [203] is not impeded by similar warnings as these technologies are associated with similar vulnerabilities. In the following, we will provide a description of the current browser war opponents and how their evolution has affected web applications. The two opponents are Mozilla's [275] Firefox [273], and MS's IE.

Internet Explorer

IE is one of the oldest browsers still prevalent today. First developed and released as a MS Windows 95 [91] upgrade during July of 1995, IE included built-in support for dial-up networking as well as the Transmission Control Protocol (TCP) [45] and the IP protocols which are key technologies used for connecting to the Internet [74]. Proof of this browser's significant stature is based upon its involvement in both of the major browser wars. Due to prevalence of the MS Windows operating system in corporate environments, IE is the lead product considered by web developers during the development of any web application. It also provides a dangerously viable target for malicious parties who wish to affect a large number of users. The more users a product has, the more likely attackers are to generate an exploit for that product. Even with the latest version of the browser, which is currently IE Version 8, updates are being issued to minimize the impact of vulnerabilities. One such vulnerability example is a Hyper-Text Markup Language (HTML) sanitization bypass weakness. IE 8 includes a method designed to sanitize executable script constructs from HTML. Attackers can bypass this protection to allow script code to execute on the client. One avenue of attack would be to insert the script within a "`postMessage`" call [340]. The attack requires the malicious user to get the victim to click on a link for this attack to succeed. If submitted to an application running on an AWS base, the HTTP `POST` message will be visible within the log file. It is this type of log file event that our tool can detect as long as the logging capabilities of the server maintains the requisite information.

Firefox

Within the current browser war, the burgeoning browser contender is Mozilla corporation's open source project called Firefox. In addition to supporting the Web standards such as HTML [322], eXtensible Hyper-Text Markup Language (XHTML) [307], Cascading Style Sheets (CSS) [36], Document Object Model (DOM)s 1 or 2 [17] and eXtensible Markup Language (XML) [38, 148]; and common proprietary plug-ins such as Java, Flash [220] and Acrobat Reader [148, 219], Mozilla has great support for the millions of non-standard web pages that exist on the Internet today. Mozilla's "quirks" and "almost standard" modes ensure that even buggy web pages display quickly and correctly [321]. Complying with the standards of a web technology enables only a certain degree of control. Web technology standards were not designed to be secure. To add to the already insecure design, compliance with non-standard controls makes the applications inherently more vulnerable.

Unknown functionalities that could be hidden within a "buggy" webpage can prove to be malicious. The capabilities provided by additional plugins or add-ons to the Firefox program are undeniably useful for a developer, however with these tools, a malicious user can also determine an effective method of attack. For example, a Firefox add-on developed with the intent to "spider" a website could retrieve not only identifiable information about a target, but also information that indicates a vulnerability is present. Machine automated user agents that autonomously wander the web issuing HTTP transactions and fetching content are known as spiders or web robots. These spiders generally wander the web to build useful archives of web content such as a search engine's database or a product catalog for a comparison shopping robot [109]. Another form of spidering that performs extensive server directory traversal is a web development application called Webscarab; this application may be retrieved from [150]. A Firefox plugin could provide functionality for viewing Flash animations. Unfortunately, additional capabilities such as this one would open up Firefox to any vulnerabilities currently affecting that functionality. One such example is the case of Actionscript invoking a malicious Actionscript-based function or invoking a malicious function that executes JS without the browser or the user's knowledge [41]. This is one example of a Cross Site Scripting (XSS) [190] attack. A generalized scenario of a XSS attack is depicted in Figure 3. More details on this attack and others are presented in Chapter 4. For more information about Firefox add-ons or plugins please refer to [272] or [274].

Opera

The ability to effectively disseminate information is the Internet's greatest asset. As more and more devices are being brought onto the Internet, the choice of interface to that information is very important. The Opera [18] web browser has assumed a leading role for use in popular "smart" devices such as mobile phones, Nintendo [97] DS [98] and Wii [99] systems, Personal Digital Assistant (PDA)s, and is also used in interactive televisions [56]. As browsers are ported to these different devices and since the expansion of any product to a new sphere may introduce new vulnerabilities, it is imperative that the security of these browsers is enhanced. With the current version of the Opera browser, which is version 10.62, there exist multiple cross-domain and address bar Uniform Resource Locator (URL) [265] spoofing vulnerabilities that have not been patched [341]. With the wide variety of browser choice, the capability to correlate both the browser history or browser log of a victim or confiscated attacker machine, with the server log could be of great assistance to any investigation. Especially in the case where the intended victim machine end point was actually the

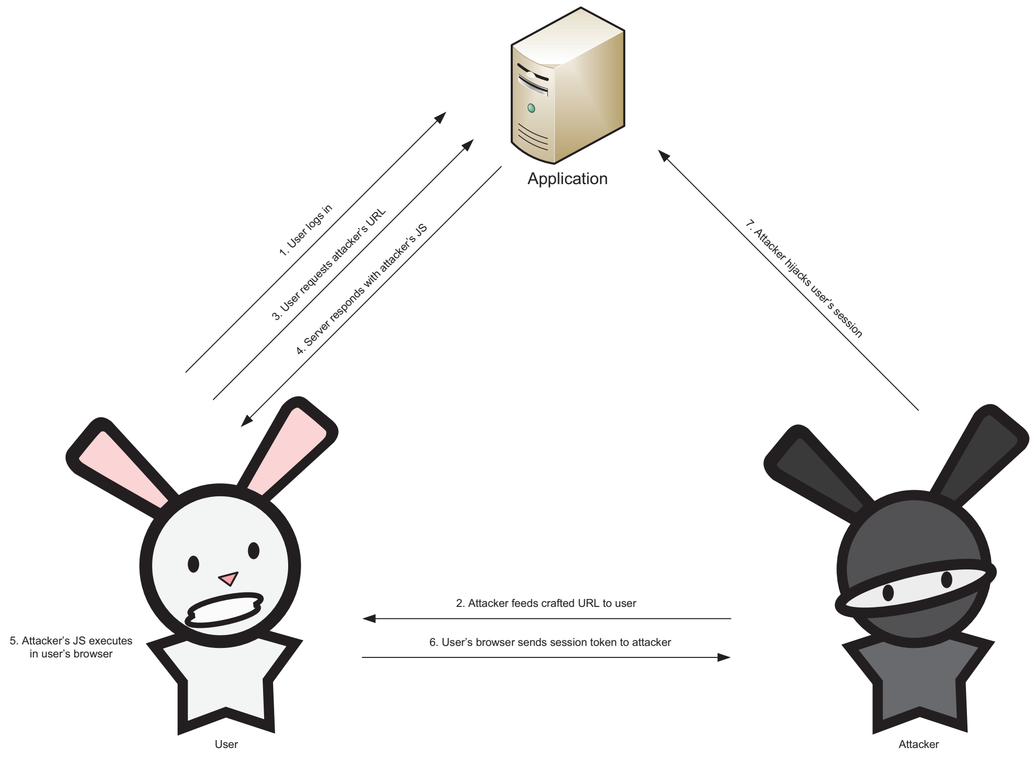

Application
1. User logs in
3. User requests attacker's URL
4. Server responds with attacker's JS
7. Attacker hijacks user's session
2. Attacker feeds crafted URL to user
6. User's browser sends session token to attacker
5. Attacker's JS executes in user's browser
User
Attacker

Figure 3: Cross Site Scripting Scenario [360]

server, rendering the client victim machine as only a means of accessing the intended victim.

2.2.2 Server Side

The web server application is setup to listen for and processs information requests. The purpose of a web server is to translate a URL either into a fileneame and transmit the requested file to the origin of the request, or into a program name, then execute that program with specified parameters if applicable, and send the results back [249]. The basic transactions that comprise the Internet include the abilities of sending and receiving data [109]. The browser and server applications provide an interaction platform for these transactions.

Apache Web Server

The AWS is the target web service chosen for our implementation. It is released as an open source licensing software project by the Apache Software Foundation. We chose this implementation because it provides comprehensive documentation, an extensible platform with a large library of freely available modules, as well as a completely configurable logging structure. Figure 4 describes the relationship between the service architecture, web applications and the filesystem structure.

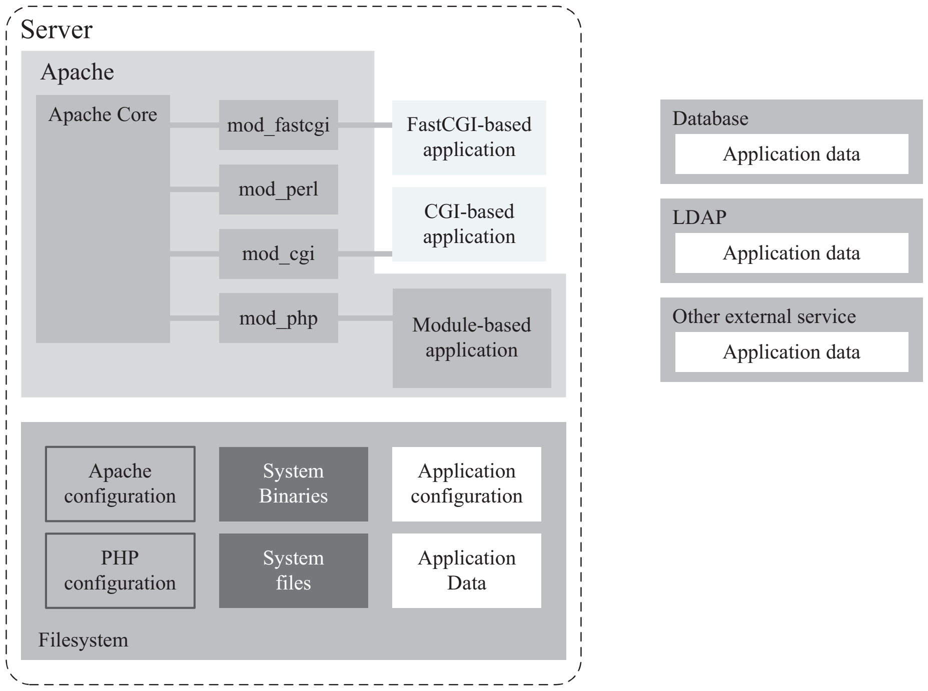


Figure 4: Web Architecture: From Apache Viewpoint [329]

With respect to security, the AWS has many configuration options that should be explicitly set to harden the installation. The default settings ensure the service gets up and running but can compromise security. Self inflicted attacks stem from common configuration mistakes, which enable attackers to easily compromise a publicly facing system. Due to the vast differences in nature for each server instantiation, there is no readily available solution that covers all the necessary configuration details required to fully secure a server subjected to the public Internet. Programmers make mistakes, and attacks will continue to occur as no system is ever truly secure. This reveals the need for forensic responses to these incidents.

The AWS is extended by the addition of modules at compile time. Depending on the module, its functionality may enhance or detract from the security of a given server instance. Those modules used within hardened installations can include `mod_evasive` [394], `mod_chroot` [202], and `mod_security` [328]. The `mod_evasive` module enables the AWS to take evasive actions in the event of a DoS [199], Distributed Denial Of Service (DDoS) [304] or a "brute force" [24] attack. It can also be used as a detection tool and configured to talk to `ipchains` [26], firewalls or routers [394]. The `mod_chroot` module enables the AWS process and any child processes to view the file system such that the apparent root directory is not the real root directory but one of its descendants [177]. This functionality is similar to the system calls `chroot()` [118] or `jail()` [234] on a Unix/POSIX compliant system. A `chroot` on such a system restricts the functionality available to a running process by only allowing access to those files within the "jail" environment. The AWS's functionality of performing as a reverse proxy is made possible due to the `mod_security` module. In the open source world, `mod_security` is an embeddable web application protection engine, also known as an application gateway appliance, that has the cababilities to perform fine grain filtration of HTTP requests and responses, normalize paths and parameters to fight evasion techniques, perform extensive audit logging, and filter Hyper Text Transfer Protocol Secure (HTTPS) [325] data [28]. It works as an AWS module installed together with `mod_proxy` [146] and other supporting modules on a separate network device in the reverse proxy mode of operation [329]. The last module considered in this introduction related to log forensics supplied within the Apache module library is `mod_log_forensics` [145] or `mod_forensics` (deprecated version). Enabling the `mod_log_forensics` module provides an administrator with extensive logging. The purpose of the `mod_log_forensics` module is to reveal request events that make the server crash. It generates a special log file where requests are logged twice: once at the beginning and once at the end of the request [329]. This enables an investigator or administrator to figure out which request occurred before the server crashed. Since upon crashing the service would be unable to log the second request event entry.

One of the first places an intelligent attacker gravitates towards are the log files of the intended victim machine. The default location of the two AWS log files, on a machine running a Unix/POSIX [193] compliant operating system will be found in `/usr/local/apache/logs`. Other common locations place these files in `/var/log` and `/usr/adm`. For the AWS the location can also be administrator specified within `CustomLog` directive [327]. Not only would this be useful to attempt to hide their intrusion; access to the server's logs can potentially yield invaluable information to an attacker by revealing configuration details, usernames, client data, misconfigurations and problematic scripts [382]. If either of the log files or the directory that they are in are writable by the AWS priviledged user, an attacker can use them to cause serious damage to a system. One direct method of accomplishing this would be to generate a symbolic link from a log file to an important system file, which would then be overwritten with logging information. Another offshoot of this would be to glean sensitive data such as usernames, proxy and server configuration information, and use this to launch other attacks. If anyone other than root, administrator or operator can alter logs, they can destroy important evidence of attacks [15]. This example proves the log file directory must remain secure and ensure permissions grant only an administrative user write capability [382]. Concerning log file analysis; although there is an enormous amount of information in the log files the data is not useful in raw form. Once analyzed and organized into a proper format, statistical information such as an estimation of the number of visitors to the site, what was most often requested, what was not requested, the amount of time individuals spent at the site, the referral information, or information that directed them to the site, high traffic hours and most importantly what was broken, what was not viewed, or what was supposed to be inaccessible to the general public can provide incident response evidence to an investigator. The problem with log file analysis is not due to programmatic error, but to the protocol that is fundamental to transactions over the Internet: the anonymous stateless HTTP protocol [327]. For instance, even origin information recorded in log file events can be suspect. To minimize bandwidth usage, Internet Service Provider (ISP)s such as AOL [221] proxy all requests originating from their base network. This can result in event entries that contain the proxy as the originating address in representation of what is mirrored at the proxy location. Requests from the user network are then directed to the proxy, without actually reaching the original service and therefore not generating events from within that network. Proxies are used to relay static information that is refreshed and updated at intervals. Any interactive components or dynamic content such as the results from a Common Gateway Interface (CGI) [332] program, which web applications are inclined to include, require direction from and are tailored to individual users, and therefore pass through the proxy to result in event generation. Proxies must be taken into consideration during the reconstruction of a forensic incident timeline. By default, the AWS generates two log files. Viewing the normal operation of the AWS within the `access_log` and `error_log` files regularly should enable one to determine if there exists a problem through anomaly detection. The question posed would become how can we automate this anomaly detection for a forensic investigation? This dissertation both proposes as well as implements an effective methodology to achieve this end. This is explained further in Chapter 5. The `error_log` file contains events generated concerning diagnostic information and will record any error messages encountered during the processing of requests [319]. The `access_log` file records all client requests made to the server [330].

Examples of diagnostic information that is logged to the `error_log` include process startup and shutdown messages, debugging data from CGI scripts, standard informational messages, critical event data and errors that occurred during request serving, which would generate status codes between 400 and 503 [249]. While the `error_log` contains all process related diagnostics, `access_log`s record all individual requests. The `error_log` events that record requests generating status codes between 400 and 503 provide a certain measure of redundancy for the AWS's logging functionality. Events recorded to the `error_log` file cannot be externally generated or modifed. However, the `loglevel` directive within the AWS's configuration enables an administrator to specify event granularity. This can ensure both the retention of pertinent information to an investigation, and the elimination of excess information, which simplifies analysis. These directives are outlined in Table 1.

Table 1: Apache `error_log` `loglevel` Directives [382]

Directive	Description
`EMERG`	System unusuable - Only process information such as startup and shutdown
`ALERT`	Immediate action required - Immediate action is necessary for proper operation
`CRIT`	Critical error - Events contain critical condition information
`ERROR`	Noncritical error - Services status including improper configuration errors
`WARN`	Warning - Non-threatening service problems, which may require attention
`NOTICE`	Normal but Significant - Normal events, which may need to be evaluated
`INFO`	Informational - Includes additional informational events or suggestions, developer aid for conflict resolution
`DEBUG`	Debug level - Logs normal events and data related to debugging

Directive event information is compounding, as each level will report all events respective to those preceeding their place in the table. The directive should be set at the lowest level possible to ensure events get logged, but not so low that it overloads the processing capabilities of the server itself. This setting used in conjunction with effective log management practices such as log rotation, ensures the retention of all forensic evidence. The format of events generated for the `error_log` file is structured [214]:

```
[ Day of Week / Month / Day / HH:MM:SS / Year ] [ LogLevel ]
[ Hostname : Location (Originating Ip Address) ] [ Error Message ]
```

The example below consists of two events, the first is an error event caused by the AWS startup notice. The second event reveals that the service is unable to find the `\htdocs` directory:

```
[Sun May 17 06:25:04 2009] [notice] Apache/2.2.4 (Ubuntu) DAV/2 SVN/1.4.4
mod_python/3.3.1 Python/2.5.1 PHP/5.2.3-1ubuntu6.4 mod_ssl/2.2.4
OpenSSL/0.9.8e mod_perl/2.0.2 Perl/v5.8.8 configured -- resuming normal
operations
[Sun May 17 06:25:04 2009] [error] [client 127.0.0.1] File does not exist:
/htdocs
```

The information recorded within the `error_log` holds potential correlation advantages to a forensic investigation. This is due to the fact that error events generated by requests may contain additional information than that recorded in the `access_log`. For example to forensically determine whether an attack based on a specific module or the AWS version could be possible, correlation of the version numbers within the `error_log` along with the attack string would be useful to an investigator. The more verbose the `error_log` directive is set to, the more likely development debug suggestions may offer additional information to an investigator.

Request logs are often examined in detail to determine individual actions logged for a specific request. However, these logs are so extensive that for the determination of client patterns, trends and preferences they must be viewed in summary. Scripting languages such as Perl [308] or Python [151] are very useful for summarizing the AWS request logs [25]. Our analysis focuses on the `access_log` due to the nature of the data maintained by the default configuration that specifies events captured to this log file. Every successful or attempted file access results in a new event added to the access_log. The standard request logging, provided by the mod_log_config [158] module, is designed to capture information about all client requests and also to store errors generated by the server or other processes similar to CGI scripts that it spawns [25]. Client interactions in the form of requests contain information such as attack strings or origin IP addresses, which are of high forensic value to an investigator.

The Common Log Format (CLF) as defined by the World Wide Web Consortium (W3C) is documented at [381]. It is known as the default log format used by most Unix-based web servers. It is also the format used by the AWS's access_log. Therefore the event format for the access_log file is structured as shown below [205, 25]:

```
[remotehost] [identd (or RFC931)] [authuser] [date] [request URL] [status]
[bytes]
```

Each field of a CLF log event entry is delimited or set apart from surrounding characters by a space character [25]. The remotehost is the IP address of the client that sent the request or the client's fully qualified hostname if HostNameLookups is enabled [25]. Hostname lookups can be enabled within the AWS's configuration when the directive is set as follows:

```
HostNameLookups on
```

However this lookup requires extra processing overhead and can cause performance issues. Therefore, this is suitably performed by many offline analysis programs after log rotation. The second field of the access_log event contains the identity of the visitor in that location such as an email address or other unique identifier. If the identd [228] protocol is used to verify a client's identity and the IdentityCheck directive is activated [382], the identity information is formed by the client's identd response. The identd protocol performs user identification as defined by [228], in which a machine is queried for the identity of the user who owns a process that initiated a TCP/IP connection. The checking performed by identd is disabled by default in the AWS. In such cases, this information may be provided directly by the browser. However, this feature was removed from most browsers when spammers would collect the email addresses and send them unsolicited email [25]. Due to these reasons, it is not unusual to have this field filled by a hyphen character as a placeholder. The authuser field contains client provided username credential information. This field will only be populated if the client requests a protected document that requires a user ID and password [382]. Basic AWS authentication is provided through the use of a .htaccess file. The date field provides the date and time of the request, enclosed in square brackets with [DD/MMM/YYYY:HH:MM:SS Timezone] format. A client's request string is recorded in an event entry enclosed in quotes and omits the leading `http://servername` portion of the URL [25]. The field entries of the request string include descriptions of the `METHOD`, `RESOURCE`, and `PROTOCOL` for each request. According to RFC 2616, the eight methods that are defined for HTTP Version 1.1 include `GET`, `OPTIONS`, `POST`, `HEAD`, `PUT`, `DELETE`, `TRACE` and `CONNECT` [131]. The `RESOURCE` portion of the request is the actual document, file or URL that was requested from the server. The `PROTOCOL` will usually be defined as HTTP followed by a version number [327]. The `status` field consists of the three digit status code returned to the client. The status code indicates the success of a request or if a problem was encountered during processing. A complete listing of the status codes specified by the AWS is listed in Table 2. Status codes can be used as correlation information in conjunction with a verified attack request attempt to determine whether the attack was successful. The last piece of information, the `bytes` field is the total number of bytes that were returned in the body of the response to the client [327]. This value does not include the amount of bytes included in any headers that are returned [382]. Deviations from the intended byte counts can indicate malicious requests.

Table 2: `access_log` Request Status Codes[327]

Series	Code	Translation
		100 Series - Informational
100	100	Continue: The client should continue with the request.
	101	Switching protocols: The server is willing to comply with the clients request to upgrade protocols.
		200 Series - Successful
200	200	OK: The request was successfully completed and occurred without error.
	201	Resource created: The resource was successfully created. A `POST` command was issued and satisfied successfully without event.
	202	Accepted: The request has been accepted for processing, but the processing has not been completed.

Table 2: `access_log` Request Status Codes[327]

Series	Code	Translation
	203	Nonauthoritative information: The information is not the definitive set as available from the origin server, but has been gathered from a local or third-party copy. The server could only partially satisfy the client's request.
	204	No content: The request was fulfilled, but no content needs to be returned.
	205	Reset content: The request has been fulfilled, and the client should reset the document view that caused the request to be sent. For example, reset the contents of an HTML form so that the user can enter new information into that form.
	206	Partial content: The partial `GET` request has been completed. This will be in response to a `GET` request that included a range header, requesting only a portion of the resource.
300 Series - Redirection		
300	300	Multiple choices: The requested resource can be fulfilled with any one of several choices.
	301	Moved permanently: The requested resource has been permanently moved to a new location.
	302	Found: The resource is temporarily located somewhere else, but the client should continue to use the same URL in the future.
	303	See other: Usually the same as a 302. The response to the requested URL can be found at another location and should be retrieved from there.
	304	Not modified: The document has not been modified since the specified date.
	305	Use proxy: The requested resource must be requested through the specified proxy, which is sent in the Location header.
	306	Unused

Table 2: `access_log` Request Status Codes[327]

Series	Code	Translation
	307	Temporary redirect: The resource has temporarily moved to a new location, and the client should repeat the request using that new location.
400 Series - Client error		
400	400	Bad request: The request was not understood by the server. (Malformed request)
	401	Unauthorized: The request requires user authentication. This response is accompanied by a request for the necessary credentials.
	402	Payment required: Not yet used.
	403	Forbidden: The request was understood, but is being refused.
	404	Not found: The requested resource could not be located.
	405	Method not allowed: The method used is not one of the methods permitted for the requested resource.
	406	Not acceptable: The requested resource is only available in representations which the client has indicated are not acceptable.
	407	Proxy authentication required: Similar to 401, but indicates that a proxy server requires authentication.
	408	Request timeout: The client did not produce a request in the time that the server was willing to wait.
	409	Conflict: The request could not be completed because of a conflict.
	410	Gone: The resource is no longer available, and there is no known forwarding address.
	411	Length required: The server will not accept the request without a `Status-Length` header.
	412	Precondition Failed: A precondition specifies in the request header evaluated is false.
	413	Request entity too large: The request was larger than the server was willing or able to process.

Table 2: access_log Request Status Codes[327]

Series	Code	Translation
	414	Request Uniform Resource Indentifier (URI) [265] too long: The request URI is longer than the server is willing to interpret. Note that this is not the same as 413, which refers to the entire request entity, including headers.
	415	Unsupported Media Type: The request is in a format not supported by the requested resource for the requested method.
	416	Request range not satisfiable: The client request included a Range specifier, which does not specify a valid range for the requested resource. For example, it requests a byte-range that extends past the size of the requested file.
	417	Expectation failed: The expectation expressed in the Expect request header could not be met by the server.
500 Series - Server Error		
500	500	Internal server error: The server encountered an unexpected condition that prevented it from fulfilling the request.
	501	Not implemented: The server does not support the functionality required to fulfill the request.
	502	Bad gateway: While acting as a gateway or proxy, the server received an invalid request.
	503	Service unavailable: The server is currently unavailable.
	504	Gateway timeout: When acting as a gateway or proxy, the server did not receive a timely response from the upstream server.
	505	HTTP version not supported: The server does not support the HTTP protocol that was specified in the request.

The extensive reconfiguration ability provided by the AWS for the access_log can be compounded by additional module functionality. The LogFormat [159] directive defines the actual format of the log file. All of the available variables that are defined in the modules mod_log_config and mod_logio [160] are shown in Table 3. The custom log format turned out to be such a good idea that even the common format was reimplemented as a custom log file format as shown [327].

```
LogFormat %h %l %u %t \ %r\ %>s %b common
```

Although advantageous to a system administrator, for forensic application developers this configurability is an issue, as the tools that they develop should conform to a variety of user generated log file types. The administrator can change the configuration to add additional detail as events are recorded. One attempt to standardize a modified CLF log is the Extended Logfile Format (ELF) [198, 25]. The ELF is referred to by the nickname “combined” by the AWS within the `LogFormat` directive.

```
LogFormat %h %l %u %t \ %r\ %>s %b \ %{Referer}i\ \%{User-Agent}i\
combined
```

The ELF adds two fields to the CLF as instantiated by the above directive is shown below.

```
[remotehost] [identd (or RFC931)] [authuser] [date] [request URL] [status]
[bytes] [referer] [agent]
```

Two example `access_log` entries are shown below.

```
91.197.197.1 - - [29/Jun/2008:16:43:34 -0400] "GET /w00tw00t.at.ISC.SANS.DFind:)
HTTP/1.1" 400 343 "-" "-"
209.91.158.189 - - [30/Jun/2008:06:36:51 -0400] "GET / HTTP/1.1" 200 923 "-"
"Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 1.1.4322; .NET CLR
2.0.50727; .NET CLR 3.0.04506.30)"
```

First introduced with National Center for Supercomputing Applications (NCSA) [287] `httpd` 1.4, the ELF was comprised of the standard CLF with two additional fields `referer` and `UserAgent`. The `referer` field logs the value of the referrer contained within the HTTP header if it exists. The presence of this field in a request indicates that the request was generated by clicking a link in a page from another site, and the content of this header indicates the URL of this page. The agent field indicates the browser used by the requester.

Table 3: `Access_log` Variables by `mod_log_config` and `mod_logio` [382, 327]

Variable	Definition
%...a	The remote IP address. %h is equivalent to %a if `HostnameLookups` are not enabled.
%...A	The local IP address, if the AWS is listening to more than one interface. Useful for IP-based virtual hosting.
%...B	Bytes sent, excluding HTTP headers.
%...b	Bytes sent, excluding HTTP headers, the value of the Content-Length header of the server reply. In CLF format that is a '-' rather than a '0' when no bytes are sent.
%...c	The connection status - see %X (AWS 1.3.15 onward, renamed in AWS 2).
%...D	The time the server took to process the request (AWS 2 only).
%...{Variable}e	An environment variable as defined by the server.
%...f	The filename of the document queried including its complete file path.
%...h	Remote host - this is the remote hostname. An entry here requires the `HostnameLookups` directive for reverse hostname lookups. If `HostnameLookups` has not been enabled, or if a reverse hostname lookup cannot resolve a client IP address to the hostname, the IP address of the requestor is logged [25]. In many cases this is not the requester's IP address but that of a proxy.
%...H	The protocol used to make the request. (Includes version).
%...{header_line}i:	The contents of one or more `header_line` entries, each specified within braces, taken from the request sent to the server.
%...I	Provided by `mod_logio`, if present. The total number of bytes received in the request, including the request line, all headers, and the body (if there was one). For secure connections, this is the number of bytes prior to decryption, not the number of bytes in the decrypted request. See also %O.

Table 3: `Access_log` Variables by `mod_log_config` and `mod_logio` [382, 327]

Variable	Definition
%...l	Remote username, the response to an identity request to the client. (from `identd`, if supplied) `IdentityCheck` directive must be enabled for a value to be returned.
%...m	The request method (AWS 1.3.10 onward).
%...noten	The contents of one or more note entries specified within braces, taken from another module.
%...{header_line}o	The contents of one or more `header_line` entries, specified within braces, taken from the reply of the client.
%...O	Provided by `mod_logio`, if present. The total number of bytes sent in the response, including the status line, all headers, and the body (if there was one). For secure connections, this is the number of bytes after encryption, not the number of bytes in the unencrypted request. See also %I.
%...p	The TCP port number that the client request arrived on, as defined by the `Port` or `Listen` directives.
%...P	The process ID (pid) of the child `httpd` process that serviced the request. For the AWS 2, see also the extended format.
%{pid—tid}P	Either the pid or thread ID (tid) of the AWS child process or thread that handed the request. %{pid}P is identical to %P. Use an expression such as ”%P and: %{tid}P” to log both values (For AWS 2.46 onward).
%...q	The query string (prepended with a ? if a query string exists, otherwise an empty string).
%...r	The first line of the request, containing the HTTP method. Equivalent to the combination of ”%m%U%q%H”.

Table 3: `Access_log` Variables by `mod_log_config` and `mod_logio` [382, 327]

Variable	Definition
%s...s	The HTTP status code, for example, 200. If the client request caused an internal redirect, %s will contain the status of the original request, and %>s the status of the eventual result (the status of the last request). In general, %>s is much more useful than %s, though there is no reason both cannot be logged.
%...t	Time, in standard "English format" [239]. Without a format, standard CLF time is: [Day/Month/Year:Hours:Minutes:Seconds Time Zone].
%...{format}t	The time, in the form given by format, which should be in `strf-time` [117, 264] format. This is potentially localized.
%...T	The time taken to process the request, in seconds. Used for spotting performance problems in CGI scripts.
%...u	Unauthorized remote user. A 401 response could mean that the user ID supplied and logged by the user was bogus; or the user ID is acceptable, but the password is missing or incorrect. The result here may be unreliable if the return status (%s) is 401.
%...U	The requested URL. %r also contains this value as part of the HTTP request.
%...v	The canonical `ServerName` [155] of the server serving the request, as defined by the `ServerName` directive.
%...V	The server name according to the `UseCanonicalName` [156] setting.
%%	A literal % symbol (AWS 2.44 onward).

In each case the "..." indicates an optional condition. If the condition is met, then the specified variable is displayed. If the condition is omitted and it cannot be defined through the request data, then the variable will be replaced with a "-" [327]. Other modules can instantiate additional variables to extend the number of fields in an event for the storage of extra data. The `mod_ssl` [127] module provides variables for logging Secure Sockets Layer (SSL)-related information as well as some powerful general-purpose logging features [382].

Lastly, another feature that can be implemented in the AWS is user tracking. As more ISPs utilize caches or clients utilize proxies while browsing, logs contain less user specific data. The AWS can utilize two modules with different methods of performing user tracking. Cookie tracking, implemented by the `mod_usertrack` [162] module is more successful because it remains uninhibited by the problems proxies face. URL tracking used by `mod_session` [161], where URLs sent to the client have tracking information embedded into them, is less successful for two reasons. The first reason is that earch engines are capable of finding and repeating the modified URLs. The second is that proxies are unable to cache any portion of the page if the URL is always changing [382]. As this information can be utilized forensically in a future application, it will be discussed further in Chapter 6.

2.3 Other Applications

The definition of a web application is quite broad. In this section, we will explore the different types of web applications and how they are utilized today within the modern Internet architecture. Basic HTTP architecture web applications including gateways, tunnels, agents, robots, caches and proxies will be introduced. Then a different classification of web applications that are built on these underlying network applications will be described and examples given. The best definition of this type of web application is a collection of dynamic scripts, compiled code or both, that reside on a web, or application server and potentially interact with databases and other sources of dynamic content [14]. Operating system independancy is indicative of the increased popularity of web applications. Since applications are built based on open standards, they can run on any type of operating system that the client uses without difficulty.

2.3.1 Gateway

Gateways are special web servers that act as intermediaries to other servers. There are two types of gateways; the resource gateway and the protocol gateway. Some gateways automatically convert HTTP traffic to other protocols, so HTTP clients can interface with these other applications without requiring the knowledge of the other protocol locally. This enables transparent access to that other protocol or application through an HTTP connection. The client may not be aware that it is communicating with a gateway. The gateway can speak the query language to the database or generate the dynamic content, acting as a portal where a request comes in and then a response comes out. The resource gateway which is the most common, is also called an application server. This combines the destination server and gateway into a single server. Application servers are server-side gateways that speak HTTP with the client and connect to an application program on the server side [109]. As with other web applications, a gateway may or may not implement logging. When providing access to protocols, and due to different pertinent values, the log events generated by the underlying services, if any, will not necessarily be applicable. If the gateway does provide relevant information in events, this could be an excellent correlation source.

2.3.2 Tunnels

HTTP tunneling consists of a bidirectional virtual data connection that is encapsulated within HTTP request and response headers. The tunnels are special proxies that blindly relay raw TCP or User Datagram Protocol (UDP) [313] data between two connections. They enable access to applications that speak non-HTTP protocols through HTTP applications by allowing other protocols to piggyback on top of HTTP connections. By embedding the non-HTTP traffic inside the connection, the traffic can be sent through firewalls that only allow web traffic [109]. The HTTP specification reserves the method `CONNECT` for use with a proxy that can dynamically switch to being a tunnel [131]. Not only useful for users wishing to proxy inbound, tunnels are useful to users placed behind restrictive firewalls. If Internet access is allowed through an HTTP proxy, this opens up the possibility for connections in both directions and poses a great security risk for machines running services other than web services.

A web application attacker may utilize a web tunnel in three ways [360]. One way involves the attacker sending HTTP `GET` and `CONNECT` requests to and via the web server which then functions as a proxy to connect to other servers on the Internet. This is one method of how untraceable hops are generated. These requests are visible within the logs of the servers they are made through. However, for the individual investigator tracing through all the log files on multiple servers may be impossible. Some logs might not be accessible due to unimplemented logging or uncooperative server owners. Another use of a web tunnel would attempt to connect to different IP addresses and ports within the hosting infrastructure itself. This would assist an attacker in mapping out the internal network, collecting passing information such as usernames, or content information that could be useful for more devastating attacks. Lastly, attempts to connect to common port numbers on the web server itself, by specifying the localhost IP address (127.0.0.1) as the target host in the request would yield valuable running services information to an attacker for further exploitation. For excellent working examples of how to accomplish these attacks, refer to the note of [360].

2.3.3 Agents

With regards to the HTTP, agents are client programs that make HTTP requests on the user's behalf. The current working standard [121] defines a user agent as any software that retrieves, renders, and facilitates end user interaction with web content. These can include semi-intelligent web clients such as "robots", "web crawlers", or "spiders" which make automated HTTP requests. However, the broad definition also includes any application that issues web requests such as browsers. The majority of browser functionality was explained above in Section 2.2.1 except for those programmatic add-ons or plugins that enable the browser to make automated requests similar to the following described programs. "Robots" are machine automated user agents that autonomously wander the web issuing HTTP transactions, or a specified series of requests to fetch content with minimal human supervision [109]. They are used to traverse website heirarchical link structures to fetch content and process any data they come across. Some example uses of "robots" can include "spiders" that are used to build archives of web content for a search engine, a comparison shopping robot to check on prices from different websites, or stock-graphing "robots" which scour data to build market trend charts. When a "robot" recursively follows web links, it is called a crawler or a "spider" because it "crawls" along the web created by HTML hyperlinks [109]. An appropriate log management tactic that should be employed by the administrator is to ensure the definition of a "robots.txt" file resident in the public directory of the web server. It is within this file that rules for legitimate automated use for the target site are defined. From a forensics perspective it is imperative to employ one of these files along with the terms of use for a particular site should a malicious incident arise. Automated agents can cause a multitude of requests within a small window of time. If malicious, the agent may be able to traverse secure areas or cause an interruption of service due the submission of too many requests. The automated appearance of multiple requests may be identified due to the same originating source, or by the correlation of event entries by request data. During the reconnaissance phase of an incident multiple requests by an automated agent may be the only precursor to indicate an imminent threat.

2.3.4 Caches

Internetworked devices that store copies of frequently used documents are known as web caches. When a web request reaches a cache, if a local copy of the document requested is available, it is served from the local storage instead of sending the request on to the remote server, where the data originated from. The advantages of caches include a reduction or elimination of data transfer redundancy, network bottlenecks, demand on the original server, and distance delays. In turn, these result in the avoidance of costly network charges, a page load time reduction due to the avoidance of packet level bottlenecks and colocated data as well as a faster reply from the origin server from lack of overhead. Due to these advantages, caches are able to solve the problem of flash crowds. Flash crowds are caused by simultaneous specific external events, which cause spikes in traffic flow [109]. These are dangerous when they present themselves in emergency situations, as they can cause communication breakdown. An outside world example of this would be an emergency happening on the campus of a university; the Short Message Service (SMS) emergency text messages that are sent out can take over the available bandwith of the voice network. An example attack scenario of this is extensively described in [312]. Inadvertant or deliberate DDoS attacks can also be performed on web services. Properly implemented caches can assist with these traffic loads.

2.3.5 Proxies

Proxies intercept and relay requests between one client, also known as a private proxy, or several clients known as a shared proxy, and an outside service or services. Private proxies can be run on a client machine in order to extend browser features, improve performance or host advertising [109]. During their execution, these web transaction intermediaries act as both a client to the external service and a server to its clientele. Thereby proxying client requests, and accessing the server on their behalf. The difference between a proxy and a gateway is that proxies speak the same protocol on both sides of the connection, while gateways connect different protocols. However as more functionality is implemented on commercial proxies this distinction is becoming blurred. The functionalities provided by proxies include enhancing the security of firewalls, caching, logging, filtering, access control, translation, and virus scanning [383]. Proxies can monitor and modify traffic to implement many useful web services. One such useful web service that a proxy can implement is a child filter that would enable unrestricted educational content, but deny access to sites flagged as inappropriate. Reverse proxies receive requests and initiate communication with other servers to locate requested information. They are used to improve the performance of slow web servers for common content [109]. The last example that should be noted is the use of proxies as anonymizers. These proxies provide heightened privacy and anonymity, by actively removing identifying characteristics from HTTP messages. They provide a next hop connection, so that the originating location can be from a separate geographic region, or appear to be coming from a spoofed IP address. Having anonymity on the Internet is a verifiable advantage to an attacker as originating IP addresses can be difficult if not impossible to forensically trace without sophisticated equipment or access to a number of collection points.

2.3.6 Retail Websites

More application specific deployments on Internet facing websites can be used for a variety of purposes. Prolfic applications that allow consumers to buy products online emerged. Major consumer stores such as Sears [1], Walmart[2], HBC[3], HMV[4], Blockbuster[5] and Futureshop[6] all have intricate web applications that comprise their website functionalities. Strictly online commercial websites that can offer both products or services, base their business on the security of online applications. Sites such as Amazon[7], Thinkgeek[8], Netflix[9], and Dell[10] all offer products or services, and deliver online purchase capability. Another major online retail website is the auction giant Ebay[11]. The variety of technologies that enable the existance of these applications are numerous. The protocols used to communicate across the world wide web may be standardized, but individual developer's programming methods and choice of development technologies can vary widely. Web applications of this complexity are comprised of new technologies to keep up with the current face of business requirements. Any new technology will have risks associated with using it, as developers and administrators battle unknown vulnerabilities and become more familiar with the functionality that the technology will provide. The broad spectrum of technologies associated with these applications melded with the fact that they comprise a source of revenue from these companies provide a solid basis for the need to be able to forensically backtrace any incident that may occur.

[1]www.sears.ca

[2]www.walmart.com

[3]www.thebay.ca

[4]www.hmv.ca

[5]www.blockbuster.ca

[6]www.futureshop.ca

[7]www.amazon.ca

[8]www.thinkgeek.com

[9]www.netflix.ca

[10]www.dell.com

[11]www.ebay.com

2.3.7 Banking and E-Commerce

Online payment systems for most companies will connect to trusted third parties. Two of the most well respected and recognized companies associated with online payment systems are Verisign[12] and Paypal[13]. Verisign generates and signs valid SSL [119] certificates for company use with online applications. Paypal provides a secure online banking environment for clients to associate a payment with a Paypal account. Real life bank branches have also automated and generated web applications that enable their customers to do banking online. Institutions such as Royal Bank[14], TD Canada Trust[15], and Scotiabank[16] all have public facing web applications. Another financial institution that has few branches and almost solely relies upon Internet banking is ING Direct[17]. One of the first things noticed upon visiting the site is a big notice posting that ING Direct is actively trying to prevent phishing [232] attacks. Prevention of attacks and incident response for financial institutions is imperative as their business relies on the trust of their clients. Even the indication that an attack has been successfully executed against one of these institutions causes a loss in revenue.

2.3.8 Online E-Mail, Social Networking, Web Logs, and Interactive Information Sites

Web applications can also provide web facing access to electronic mail. Complex and multifaceted application collections such as the open source Horde Application Framework [204] are utilized on a big scale by major institutions. The three major public facing web applications that implement online access to email include MS Hotmail, now known as Windows Live[18], Google mail[19], and Yahoo mail[20]. When viewing electronic mail forensically, interactions between individuals can indicate both common groupings and may include pertinent incident information such as intended attack targets. Correlation between the email service logs and significant timeline events may produce additional corroborating evidence. Should an email web application contain vulnerabilities, this would open up avenues of attack such as the dissemination of confidential information or masquerading as legitimate users.

[12]www.verisign.com

[13]www.paypal.com

[14]www.rbcroyalbank.com

[15]www.tdcanadatrust.ca

[16]www.scotiabank.com

[17]www.ingdirect.ca

[18]www.hotmail.com

[19]www.gmail.com

[20]mail.yahoo.com

Social networking profiles on sites such as Myspace[21] and Facebook[22] are both supported by diverse web applications, driven by multiple technologies. For Myspace, these technologies include the Representational State Transfer (REST) [133] style of architecture for social media platforms. Facebook employs a LAMP as its base network level architecture. Other technologies in use by Facebook include Memcached [120], C++ [358], Java, Python, Erlang [129], Thrift [141], Scribe [9], Cassandra [154] and Hiphop [70] for PHP [40]. For more information on how these technologies are used by the Facebook web application, refer to [40]. This demonstrates the complexity of a modern web application, and with multiple interactions new vulnerabilites develop. The ability to log data based on individual technologies may offer guidance to developers and evidentiary material to investigators with respect to the relationships between technologies. Twitter[23] is a social networking and microblogging service, where users send "tweets" which are short messages no longer than 160 characters in length. It has a very large user base with over 175 million users [277]. In April of 2009, an attack worm uploaded and propagated by a XSS vulnerability affected approximately 100 accounts and over 10,000 "tweets" that could spread the worm were deleted [49]. Yet again in September of 2010, a website update reintroduced a flaw that meant JS could be injected into "tweets" [254]. Examples such as these demonstrate that web application vulnerabilities are a serious concern for developers. They need to be addressed for all web technologies and their underlying infrastructures. After incidents such as these, investigators can employ forensic techniques to determine the extent and construct a timeline of events that led to the execution of this malicious attack.

Currently the concept of citizen journalism for people involved in the information technology sector is rampant. Employers tend to seek out and label those job candidates who maintain an online journal with technology related entries known as a web log or blog. Blogs can also be maintained as personal, political, or other subject specific diaries. The information contained within these online diaries can get one hired, but there have been cases where it can also impact negatively on an individuals situation. An example in London was described here [222]. Two websites that offer this type of service include Blogger[24] and Livejournal[25]. They have both implemented and maintain web applications as their interface for their websites. In each case there is a login structure, and depending on the application, specific customizations can be made by individual users. When designing these applications developers must always keep in mind that the application's functionality

[21]www.myspace.com

[22]www.facebook.com

[23]www.twitter.com

[24]www.blogger.com

[25]www.livejournal.com

is determined by user input. The application must respond to badly designed or malicious input in a secure manner.

Interactive information sites such as online video or radio, search engines or wiki's are all examples of complex web applications. Online video sites include Hulu[26], Youtube[27], as well as the conversion of common television channels to produce online media. In Canada, three great examples of this conversion consist of the CTV Television Network [28], the Global Television Network[29], and the Space Network[30]. A major radio station in Toronto known as CHUM FM 104.5[31] has a web application that enables listeners to see the lineup and access previously played media according to date and time, along with listening to live streaming audio over the Internet. Search engines continually evolve their web applications to add new functionalities. For example Google, [32] introduced a new search enhancement called "Google Instant" which predicts the search query and starts showing search results as text is typed. Wiki's are defined as a piece of server software that allows users to freely create and edit web page content using any browser [106]. Today, founded by Wikimedia, the largest collaborative wiki is known as Wikipedia which resides here [55]. Its existence as a charitable organization, is funded primarily through donations. The underlying web application technology that drives Wikipedia is MediaWiki which is a free software wiki package available here [54].

2.3.9 Corporate Developed Intranet or Vendor Supplied Applications

Many corporations devise their own intranet web applications that can be delivered using HTTP. This is due to the interoperability with many different operating systems, as well as the excellent logging capabilities on an internal network for the web service. We have already shown that using the AWS, all requests are logged, and there are user tracking modules that pinpoint within an intranet all of the interworkings of individual employees. Both the `HostnameLookup` directive can be enabled server side and client side, as well as the `identd` service to provide even greater tracking capabilities. Both of these are not applicable to the Internet as a whole, but can be useful to a company internally. Something that may need to be explored in the future is how to deal with cloned or rogue employee machines, and how to forensically determine that a breach of this nature has occurred.

[26]www.hulu.com

[27]www.youtube.com

[28]watch.ctv.ca

[29]www.globaltv.com

[30]watch.spacecast.com

[31]www.chumfm.com

[32]www.google.com

As an end note to this section, although applications may not be public facing, any vulnerabilities may be leveraged from inside of a network. Two examples of vendor supplied product specific web applications that have been reported to contain vulnerabilities include the Hewlett Packard (HP) [71] Printer Management console [299], which includes a XSS attack vector and the Help application provided by MS on both Windows XP [94], and Server 2003 [93] platforms [297], which incorrectly handle escape sequences. These vulnerabilities provide examples of how manufacturers can postpone patches until the next product release and how latent vulnerabilities can still be used by attackers. Unless a patch is provided, or an administrator has another method of preventing the attack, the application remains vulnerable. Within the last section, the purpose of detailing individual websites was to draw the readers attention to problematic areas. It was also to provide documentation as to the severity of the current security of currently published applications. XSS along with other variant attacks performed on web applications can result in severe damages. What makes the attacks described above especially heinous is that they are performed along with what seems to be a valid session as opposed to the scenario of a phishing attack, in which the client may be able to discern the lack of session state. Unless some other functionality that the client is made aware of is incorporated into the attack, the client may not realize that they have been compromised.

2.4 Regular Expressions

Regular expressions, also referred to as `Regex` or `Regexp`, provide a concise and flexible means to match or match and replace strings of text. The text is comprised of particular characters, words or patterns. Strings known as patterns are created using the `Regex` language determined by and built into other languages and products. Since patterns are descriptions of text, not specifications, this is extremely advantageous to a programmer that only knows the traits of the pattern target. Regular expression patterns allow for a characteristics description of the item sought without specifying the item explicitly. They also enhance the built in search capability of a language which may not be able to exclude or distinguish certain matches based on respective criteria. A matcher determines the results of applying a pattern. Typical use cases for `Regex`s include textual validation, searching, modification, and replacement. Regular expressions define a concise and flexible way for comparing and matching string objects. An object string may be composed of patterns that include particular letters, words, symbols, or punctuation characters or a combination thereof. A `Regex` is written in a formal language that can be interpreted by a `Regex` processor.

Here we list some simple examples of `Regex`s and their represented text:

- `car` : the appearance of the letter sequence '`c`', '`a`', '`r`' in any context, such as "`car`", "`cartoon`", "`bicarbonate`", "`a car`", "`12_car|*&$`", etc;
- `(his|her)\s(book)` : the word "`his`" or "`her`" followed by a white space (e.g., space character), followed by the word "`book`";
- `[0-9]+[\w]{2,6}` : one or more digits (from 0 to 9) followed by a number of word characters (e.g., letters), the exact number of characters is between 2 and 6 in this case.

In the `Regex` string, some characters are designated with special meanings (in addition to their original ones), which are known as `meta-characters`. In this section, we detail a range of particular combinations of symbols as `control sequences`, such as "negative lookbehind" (`?<!`). They will guide the processor to perform different kinds of operations while parsing the provided `Regex` string. The related discussion will be further explored in Section 2.4.2. In the next section, we will first introduce the syntax of `Regex` language.

2.4.1 Syntax

This section provides a comprehensive view of common `Regex` syntax. The regular expressions we collected which defined attacks within the log files were adapted from multiple sources. Since `Regex`s depend on the implementation by the language or product developer, there may be variations or extensions for certain metacharacters or functions. Good resources for language or product specific `Regex` directives include the man pages for `grep` [172], `egrep` [172], `awk` [11] or `sed` [35], language definition files, or other forms of product documentation such as guides or user manuals.

Characters

Literal textual characters as defined in `Regex`s are listed within Table 4 in both their natural and American Standard Code for Information Interchange (ASCII) [130], American National Standards Institute (ANSI) [16] or Unicode [12] code states. It explains the regular expression method of matching characters and metacharacters, both their symbols and their representations.

Table 4: Regular Expression: Characters [188]

Character	Description	Example
`. (period)`	Matches any single character except the line break characters `\r` and `\n`. In most `Regex` flavors, "." can be configured to match these line break characters in addition to the regular characters it matches.	matches `x` or any other character
Any character except `[\$.|?*+()`	All characters except the listed special characters match a single instance of themselves. { and } are literal characters, unless they are part of a valid token such as the {n} quantifier.	a matches a
\followed by any of `[\^$.|?*+(){}`	A backslash escapes meta-characters to suppress their special meaning.	`\*` matches `*`
`\R...\P`	Matches the characters between `\R and \P`, suppressing the meaning of special characters.	`\R+-*/\P` matches `+-*/`
`\xFF` where `FF` are 2 hexadecimal digits	Matches the character with the specified ASCII/ANSI value, which depends on the code page used. Can be used in character classes.	`\xB5` matches μ when using the Latin-1 [243] code page.
`\n, \r` and `\t`	Match a Line Feed (`LF`) [130] character, Carriage Return (`CR`) [130] character and a tab character respectively. Can be used in character classes.	`\r\n` matches a Disk Operating System (DOS) [84] / Windows `CRLF` [130] line break.
`\a, \e,` `\f and \v`	Match a bell character `(\x07)`, escape character `(\x1B)`, form feed `(\x0C)` and vertical tab `(\x0B)` respectively. Can be used in character classes.	
`\cA` through `\cZ`	Match an ASCII character Control `+A` through Control`+Z`, equivalent to `\x01` through `\x1A`. Can be used in character classes.	`\cM\cJ` matches a DOS/Windows `CRLF` line break.
`\N{name}` or `\N{U+hhhh}`	Matches a named Unicode character.	`\N{Sigma}` or `\N{U+03A3}` matches a Unicode Σ character

Character Sets or Bracketed Expressions

The fact that `Regex`s include set theory functionality using character sets known as bracket expressions [172], shows their foundation in algebraic mathematics. Inside a bracket expression, different rules apply. Rules contained within Table 5 are only valid within bracket expressions. Rules from other tables may not be valid when contained within brackets [188].

Table 5: Regular Expression: Character Sets [188]

Character	Description	Example
`[`	Starts a set.	
Any character except `^-]\` can be added without an escape to the set	All characters except the listed special characters.	`[abc]` matches `a`, `b` or `c`
`\` followed by any of `^-]\`	Used to escape special characters	`[\^\]]` matches `^` or `]`
`-` except immediately after the opening `[`	Specifies a range of characters. (Specifies a hyphen if placed immediately after the opening `[`)	`[a-zA-Z0-9]` matches any letter or digit
`^` immediately after the opening `[`	Negates the set, causing it to match a single character not listed in the set. (Specifies a caret if placed anywhere except after the opening `[`)	`[^a-d]` matches `x` (any character except `a`, `b`, `c` or `d`)
`\d`, `\w` and `\s`	Shorthand for sets matching digits, word characters (letters, digits, and underscores), and whitespace (spaces, tabs, and line breaks). Can be used inside and outside bracket expressions.	`[\d\s]` matches a character that is a digit or whitespace
`\D`, `\W` and `\S`	Negated versions of the above. Should be used only outside sets.	`\D` matches a character that is not a digit
`[\b]`	Inside a set, `\b` is a backspace character.	`[\b\t]` matches a backspace or tab character
`\xhh`	Shorthand for sets matching hexadecimal character representations. The two hh characters represent the ASCII character in hexadecimal.	`\x40` matches an `@` sign character
`\c`	Shorthand for sets matching ASCII control characters. The second letter is uppercase A through Z to indicate Control-A through Control-Z.	`\cM` matches a carriage return just like `\r` or `\x0D`
`\0xxx`	Shorthand for sets matching octal character representations. The three xxx characters represent the ASCII character in octal.	`\0` matches a `CR` just like `\r` or `\x0D`

Positional Selections

Table 6 outlines pattern directives used for selections based on specific positional ordinances of certain characters, rather than specifying the character itself. Positional pattern instructions can be based on positions within an entire subject string as well as specific to individual words within the subject string.

Table 6: Regular Expressions: Anchors [188]

Character	Description	Example
`^` (carat)	Matches at the start of the string the `Regex` pattern is applied to. Most `Regex` flavors have an option to make the caret match after line breaks as well.	`^.` matches `a` in `abc\ndef.` Also matches `d` in "multi-line" mode.
`$` (dollar)	Matches at the end of the string the `Regex` pattern is applied to. Most `Regex` flavors have an option to make the dollar match before line breaks. Also matches before the very last line break if the string ends with a line break.	`.$` matches `f` in `abc\ndef.` Also matches `c` in "multi-line" mode.
`\A`	Matches at the start of the string the `Regex` pattern is applied to. Never matches after line breaks.	`\A.` matches `a` in `abc`
`\Z`	Matches at the end of the string the `Regex` pattern is applied to. Never matches before line breaks, except for the very last line break if the string ends with a line break.	`.\Z` matches `f` in `abc\ndef`
`\z`	Matches at the end of the string the `Regex` pattern is applied to. Never matches before line breaks.	`.\z` matches `f` in `abc\ndef`
`\b`	Matches at the position between a word character (anything matched by `\w`) and a non-word character (anything matched by `[^\w]` or `\W`) as well as at the start and/or end of the string if the first and/or last characters in the string are word characters.	`.\b` matches `c` in `abc`
`\B`	Matches at the position between two word characters (i.e the position between `\w\w`) as well as at the position between two non-word characters (i.e. `\W\W`).	`\B.\B` matches `b` in `abc`
`\<`	Matches at the beginning position of a word.	`.\<` matches `a` in `abcdef`
`\>`	Matches at the end of a word.	`.\>` matches `f` in `abcdef`
`\G`	Matches at the position where the previous match ended, or the position where the current match attempt started. Matches at the start of the string during the first match attempt.	`\G[a-z]` first matches `a`, then matches `b` and then fails to match in `ab_cd`.

Quantifiers

Quantifiers determine the number of times an expression should exist in a pattern to result in a successful match. For the quantifier to be greedy, any optional item is included in the pattern match first where possible. Being greedy also means that the largest number of optional items possible is matched before the smaller number. A lazy quantifier excludes the optional item from the match if possible. Being lazy also means that the smallest number of optional items possible is matched before any larger quantities. Quantifiers are especially important when a regular expression determines the processing of values based on the successful matches due to the method they employ when matching. It is important to be aware of each directives match order as demonstrated in Table 7.

Table 7: Regular Expressions: Quantifiers [188]

Character	Description	Example
`?` (question mark)	Makes the preceding item optional. Greedy.	`abc?` matches `ab` or `abc`
`??`	Makes the preceding item optional. This construct is often excluded from documentation because of its limited use. Lazy.	`abc??` matches `ab` or `abc`
`*` (star)	Repeats the previous item zero or more times. Greedy.	`".*"` matches `"def" "ghi"` in `abc "def" "ghi" jkl`
`*?` (lazy star)	Repeats the previous item zero or more times. Lazy.	`".*?"` matches `"def"` in `abc "def" "ghi" jkl`
`+` (plus)	Repeats the previous item once or more. Greedy.	`".+"` matches `"def" "ghi"` in `abc "def" "ghi" jkl`
`+?` (lazy plus)	Repeats the previous item once or more. Lazy.	`".+?"` matches `"def"` in `abc "def" "ghi" jkl`
`{n}` where `n` is an integer `>=` 1	Repeats the previous item exactly `n` times.	`a{3}` matches `aaa`
`{n,m}` where `n >= 0 and m >= n`	Repeats the previous item between `n` and `m` times. Greedy, so repeating m times is tried before reducing the repetition to n times.	`a{2,4}` matches `aaaa`, `aaa` or `aa`
`{n,m}?` where `n >= 0 and m >= n`	Repeats the previous item between `n` and `m` times. Lazy, so repeating `n` times is tried before increasing the repetition to `m` times.	`a{2,4}?` matches `aa`, `aaa` or `aaaa`
`{n,}` where `n >= 0`	Repeats the previous item at least `n` times. Greedy. `n` times.	`a{2,}` matches `aaaaa` in `aaaaa`
`{n,}?` where `n >= 0`	Repeats the previous item `n` or more times. Lazy.	`a{2,}?` matches `aa` in `aaaaa`

Grouping

In Table 8, a description of symbols that define groups, pipes, and backreferences is given. The logical ”or” command as defined in regular expressions is accomplished by the ”|” directive. Parentheses in `Regex`s define backreferences. Backreferences are similar to variables, and they allow patterns to refer to previous matches [138]. For example, if \1 is used, it would refer back to the first subexpression, \2 the second and so on. These are used to find specific instances of subexpressions that are repetitions in a source file.

Table 8: Pipes, Grouping or Backreferences [187]

Character	Description	Examples
| (`pipe`)	Causes the `Regex` engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options. The pipe has the lowest precedence of all operators. Use grouping to alternate only part of the `Regex`.	a) `abc|def|xyz` matches `abc`, `def` or `xyz` b)`abc(def|xyz)` matches `abcdef` or `abcxyz`
(regex)	Round brackets group the `Regex` between them. They capture the text matched by the `Regex` inside them that can be reused in a backreference, and they allow for the application of `Regex` operators to the entire grouped `Regex`.	`(abc){3}` matches `abcabcabc`. First group matches `abc`.
`(?:regex)`	Passive groups use parentheses to group the `Regex`. Regular expression operators can be applied. These do not capture anything and do not create backreferences.	`(?:abc){3}` matches `abcabcabc`. No groups.
`\1` through `\9`	Substituted with the text matched between the 1st through 9th pair of capturing parentheses. At least 9 backreferences, instantiations may include more.	`(abc|def)=\1` matches `abc=abc` or `def=def`, but not `abc=def` or `def=abc`.

Modifiers

Any modifiers can be turned on at any point in a regular expression pattern by specifying them where they should begin. They can also be turned off arbitrarily by preceeding the modifier with a minus sign to continue matching afterwards without the modifier. As can be seen in the second last row of Table 9, one can also specify selection or omission of multiple modifiers together to affect a pattern. Older `Regex` flavors may apply modifiers to the entire `Regex` itself.

Table 9: Modifiers [187]

Character	Description	Example
`(?i)`	Turn on case insensitivity for the remainder of the `Regex`.	`te(?i)st` matches `teST` but not `TEST`.
`(?e)`	Used to evaluate during a search and replace `Regex`.	
`(?s)`	Turn on ”dot matches newline” for the remainder of the `Regex`.	
`(?g)`	Turn on global matching. Used for search and replace `Regex`s, with this modifier specified, it will replace all matches not just the first instance.	
`(?m)`	Caret and dollar match after and before newlines for the remainder of the `Regex`.	
`(?U)`	Inverts the greedyness of the quantifiers so they are not greedy by default, but become greedy if followed by a ”?” [195].	
`(?x)`	Turn on free-spacing mode to ignore whitespace between `Regex` tokens, and allow # comments.	
`(?i-sm)`	Turns on the options ”i” and ”m”, and turns off ”s” for the remainder of the `Regex`.	
`(?i-sm:regex)`	Matches the `Regex` inside the span with the options ”i” and ”m” turned on, and ”s” turned off.	`(?i:te)st` matches `TEst` but not `TEST`.

Replace Directives

These define those sections of the pattern that are to be replaced, and their replacements. Examples in Table 10 are based on variable references assigned from the following match:

```
                                        12                     2
"A date is 2011-02-14 in ISO format"=~ m/\b((romantic|delicious)
 34        4 5               56 7                            76 31
|(([0-9]{4})-(1[0-2]|0[1-9])(-(3[0-1]|0[1-9]|[1-2][0-9]))?))\b/;
```

Table 10: String Replacement[187] [167] [252]

Character	Description	Examples
`$n`	Matches the nth non-passive group.	a) `$2` matches `"lmno"` in `/^(l0lcatz(lmno))$/`
	These references are to be used with backreferences defined by the parentheses `()`. Backreferences are explained in Table 8.	b)`$1` matches `"lmno"` in `/^(?:w00t)(lmno)$/`
	Remaining table cells demonstrate values from the example match given.	a)`$1` matches `"2011-02-14"`
		b)`$2` is undefined
		c)`$3` is `"2011-02-14"`
		d)`$4` is `"2011"`
		e)`$5` is `"02"`
		f)`$6` is `"-14"`
		g)`$7` is `"14"`
$\`	Reference to all contents before a matched backreference string.	$\` is assigned `A●date●is●`
$´	Reference to all contents after a matched backreference string.	$´ matches `●in●ISO●format`
$+	Reference to all contents of the last matched string.	$+ matches `-14`
$&	Reference to the entire matched string.	$& matches `2011-02-14`
$_	Reference to the entire input string.	$_ matches `A date is 2011-02-14 in ISO format`

2.4.2 Regular Expression Control Sequences

Table 11 includes more advanced regular expression directives such as backtracking, atomic grouping, posessive quantifiers, lookaround and conditional operations. When `Regex`s fail to match after a branch point, backtracking occurs. A branch point is defined where the `Regex` engine chooses between two or more options. Similar to a program's logical `fork()` [207] operation, a `Regex` engine performs a backtrack operation when it has reached the end of a logic decision branch and the pattern fails to match. The backtrack operation forces the engine to return to the origin of the branch, and to select and try another path. This is done until all logical pathways have been explored. Independent subexpressions, also referred to as atomic groups, are used to optimize and restrict resultant matching sets for `Regex`s. When a string is matched within an atomic group, it is grabbed independently and remains immutable from the rest of the pattern. Once matched, the part of the input matched is excluded from analysis by the rest of the regular expression.[1] Within

[1]Once the atomic group is matched, that part of the match is taken out of the string being analyzed. This means no other part of the outside `Regex` may match it.

the group itself, backtracking may occur to obtain a match. The group can also be backtracked over, but what defines the atomic group is the fact that the text group matched cannot be backtracked into [359]. This provides the optimization mentioned previously. When nesting quantifiers are used within a `Regex`, the engine can try redundant permutations causing unnecessary overhead.

Possessive quantifiers are independent and greedy. Their functionality is the combination of each quantifier's normal greedy counterpart plus the attributes of an atomic grouping. As demonstrated in Table 7, greedy quantifiers attempt to match all optional pattern components first before continuing the processing of the rest of the `Regex`. The atomic nature of a possessive quantifier effectively eliminates backtracking once a match has been found for the greedy quantifier "atomic group".

Forward facing lookaheads, and reverse facing lookbehinds are lookaround operations that specify positions within the subject string being analyzed. These complex anchors do not consume any particular characters or expand the match. Lookaheads match the subexpression beginning at the current position, and then testing characters to the right of it. Lookbehinds test the subexpression ending at the current position, and testing characters to the left of it. Positive lookaround operations indicate that if a match is found the result is successful. Negative lookaround operations indicate that if the match is not found, the result is successful. Only those states within the lookaround construct are available to the regular expression engine. This means that if the subexpression within the lookaround is not matched, and the `Regex` engine deems backtracking necessary, the lookaround result is failure. If the lookaround containing the subexpression was positive this means failure, and if it was negative the fact that the engine tried to backtrack returns successfully [167].

A regular expression that contains a conditional construct branch possible regular expression engine run paths. If the predicate[2] returns true, then the consequent[3] is attempted and must match for the conditional to be successful. When false is returned by the predicate subexpression, the alternative[4] is then attempted and must match for the conditional construct to be successful. A conditional construct does not have to contain an alternative portion, specified by the omission of the pipe `|` character and the `else` portion [167]. In this case, success and failure are both designated by a valid predicate and the outcome of the consequent portion. Capturing groups or the four different methods of lookaround operations can be used as predicates within conditional constructs.

[2]the "if" construct portion
[3]the "then" portion
[4]the "else" portion of the construct

For the `Regex` to enter the conditional construct with a capturing group, the conditional checks a prior successful match of the first capturing group [252]. For the `Regex` to enter the conditional construct with a lookaround operation, the lookaround operation must have already matched with success.

Table 11: Advanced Regular Expression Directives [187, 167, 252]

Character	Description	Examples
`(?>regex)`	This defines an atomic group, or independent subexpression within a `Regex`.	`x(?>\w+)x` is more efficient than `x\w+x` if the second `x` cannot be matched.
	The `Regex` `a(bc|b)c`(capturing group) matches `abcc` and `abc`.	`a(?>bc|b)c` (atomic group) matches `abcc` not `abc`
	For the string `"ab"`, the `Regex` `/a*ab/;` matches	but `/(?>a*)ab/;` doesn't match
`?+, *+, ++ and {m,n}+`	Possessive quantifiers are atomic greedy quantifiers.	`x++` is identical to `(?>x+)`
`(?=regex)`	Zero-width positive lookahead. For `one(?=two)three`, both `two` and `three` have to match at the position where the match of `one` ends.	`e(?=r)` matches the second `e` in `repaper`.
`(?!regex)`	Zero-width negative lookahead.	`e(?!r)` matches the first `e` in `repaper`.
`(?<=regex)`	Zero-width positive lookbehind.	`(?<=r)e` matches the first `e` in `repaper`.
`(?<!regex)` or `(?!=regex)`	Zero-width negative lookbehind. Matches at a position if the pattern inside the lookahead cannot be matched ending at that position.	`(?<!r)e` matches the second `e` in `repaper`.
`(?(?=regex) then|else)`	Lookahead and conditional `Regex`. Note that the lookahead is zero-width, so the "then" and "else" parts need to match and consume the part of the text matched by the lookahead as well.	`(?(?<=a)b|c)` matches the second `b` and the first `c` in `babxcac`
`(?(1)then|else)`	Conditional `Regex`.	`(a)?(?(1)b|c)` matches `ab`, the first `c` and the second `c` in `babxcac`

2.5 Summary

With the introduction of Web 2.0 technologies, by the second annual O'Reilly Web 2.0 Summit in 2004, [296] the face of the Internet officially changed. No longer were users subject to only static websites, and applications that provided only low level network functions. The new technologies now provide access to dynamic and interconnected applications with a high-level of user-directed input. Web applications are what drives the features of Web 2.0. The forensics overview introduces the importance of all log files to an investigator during the conduct of a computer incident investigation. Details of applications in the categories of client side, server side, network oriented as well as those increasing in complexity defined by the standard of being directly affected by web vulnerabilities are presented. Incorporated within the last section is an introduction to regular expressions as well as an explaination of the syntax of these highly adaptable programmatic tools. To directly answer the questions posed in the introduction to this chapter we provide these highlights. A definition of web applications are the applications that reside on a web server, which have the possibility of interacting with dynamic content from both other applications as well as end users, and can consist of static code, dynamic code or both. Web applications have evolved from simple network level scripts and code that only perform low level operations, to complex, dynamic, interconnected, user directed organisms that continually evolve. The question of the security of web applications due to the effect of rapidly emerging technologies is left as an exercise to the reader to answer based on their best judgement. The technologies that web applications are built upon were described extensively in the section about vulnerability affected applications. To test the security of a web application a penetration tester cannot limit themselves to just the application itself, as this is not what a malicious party would do. Similarly a forensic examiner cannot limit the scope of an investigation to the web application itself, but must also include the associated underlying technologies associated with the application. Web applications can be examined forensically through effective management and analysis of the log files relevant to the investigations target web application. Combining these topics together not only provides the relevant background data, but when applied can lead to the design of an effective forensic web application tool, as will be described in Chapter 5.

Chapter 3

Log Analysis Tools

This chapter aims to provide an overview of the current state of the art in log analysis. The reviewed applications with log analysis functionality are not necessarily applied to forensics directly, however they may be leveraged during an investigation to provide evidence. The three types of tools associated with log analysis discussed in this section include tools made specifically to perform log analysis, tools that provide intrusion detection or file integrity monitoring, and tools that are used to scan web applications for vulnerabilities. Current log analysis tools are integrated with either system administration monitoring systems or within forensic analysis toolkits. Most analysis functionality is rudimentary or has to be adapted manually to perform the necessary functionality. This makes the usability of the application itself and the reliability of the information output paramount. In a forensic examination, real-time monitoring logs may not be availble. Most evidence is collected and analyzed after an incident, which can be problematic if the target has not properly managed resident log files [2]. Problems arise if a compromise has occured, which concerns the integrity of the log files. The integrity cannot be definitively determined unless it has been provided by another application or the log files have been secured via other means prior to the incident. In our opinion, an excellent log management strategy for effective analysis with regards to Unix compatible operating systems as mentioned briefly in [360] is outlined in [2]. With regards to web applications, security considerations cover not only the web application itself, but also underlying technologies such as the server's operating system, the web service being used to host the site, the backend database technology, the programming language, and any additional hooks corresponding to non-standard networking components or other web applications. Web application vulnerability scanners are usually provided within penetration testing toolkits such as Backtrack [255], or OpenVAS [184].

3.1 Application Specific Log Analysis Tools

Within this section we aim to provide an overview of the current tools that perform or enhance log analysis techniques. In depth log analysis tools in this category would be classified as Security Information Event Management (SIEM) [363] technologies. SIEM is an intersection of Security Information Management (SIM) [363] which is concerned with the discovery of bad behaviour through data collection and Security Event Management (SEM) [363] which is concerned with real-time activities of network devices [262]. Differentiating these tools from those in 3.3, classification may reside more in SEM type technologies. For the sake of simplicity, default logging processes provided by the underlying operating system will be overlooked.

3.1.1 Acct and Acctsum

A process accounting service is utilized to keep a detailed audit trail of all the commands executed on a Unix-based system. Process accounting provides a log of every command executed by users including associated processing cycles and length, allocated memory, and call time [181]. The `acct` [173] package is a collection of several utilities that monitor processes including `ac` [168] , `accton` [174] , `lastcomm` [169] and `sa` [171]. The command `ac` gives user logon statistics. The kernel process account log generated by `accton` is found on Unix/POSIX-compliant operating systems and includes system-wide process accounting events. Similar to the `history` [166] command, `lastcomm` displays previously executed commands. The `sa` command summarizes and logs the output of the `lastcomm` command to a file called `savacct` or `usracct`, depending whether the command was issued on a per command or a per user basis [180]. The process logger tool `acctsum` [27] consists of two scripts that truncate and summarize the kernel process account logs to produce output in HTML [37].

Forensically, having log events of all processes associated with a time stamp is an excellent method of finding out exactly what was running at the time of an incident. This is a good source of correlation information. The generation of this log file is also above and beyond the normal logging functions of a system, and would add another layer of defence. During an attack the log file provides redundancy as in the event of an incident, in additional to the default logs generated, it too would need to be modified to hide the intentions of an attacker. Providing security through obscurity, an unsophisticated attacker may not know this logging process would be running on a target system. Pertaining to analysis, `acctsum` only provides summary statistical information to the user. During truncation, there exists the possibility of events pertinent to an investigation remaining unreported, except as part of the value of the summary statistics. As with many log analysis tools attack detection, rogue process analysis and anomaly analysis must be accomplished by the investigator. As both `acct` and `acctsum` are very specific to process accounting log files, this functionality can be utilized to integrate the summary information and truncated logs into other tools. Individually, they provide insufficient functionality for a fully integrated log analysis tool.

3.1.2 Analog

The open source web log file analyzer `Analog` [373] can accept AWS or IIS W3C formatted files as input. The analysis results produce complex graphs and report styles when used in conjunction with another tool called Report Magic [261, 373]. Statistical reporting, or the conversion of statistical information into a graphical representation forms the basic functionality of this tool. Available reports styles include a general summary, time-based reports, host, domain and organization reports, file-based, browser-based as well as user and status-based reports. The validity of information retained in these reports is only maintained when the necessary server configuration is performed [372]. A good overview of information for the sole purpose of reporting on the web service alone is formulated in the results of this tools analysis. The tools design did not specify that it is a business analytics tool and would require extra configuration on the part of the analyst to provide a concise account detailing information that would be beneficial to both marketers and forensic investigators.

3.1.3 Anteater

The open source Message Transfer Agent (MTA) [103] log file analyzer `anteater` [128] is written entirely in the C++ programming language. The two types of log files that can be analyzed by `anteater` include those produced by `sendmail` [343] and `postfix` [379] agents [128]. The documentation for this project is not sufficiently updated to give an entire overview of what this tool can accomplish. From the information gathered, however, it was determined that the output can be formated into HTML, or any transposition that can be made utilizing command line syntax. Screenshots of different scenarios of this utility are shown on the project homepage [128]. These detail the eight analysis scenarios that `anteater` can perform, which include local traffic and summary of received bytes per count, the ratio of local to external traffic distribution, total byte counts of mail to external recipients from local accounts, the total number of emails sent to specific addresses and summary, the total number of emails received from specfic addresses and summary, largest mail message size in bytes and emails with the most recipients. This tool provides statistical summary data of all mail transferred to and from the host.

The `anteater` tool is another example of a log file analyzer, that does not perform comprehensive forensic log analysis, but provides insight into the processing of MTA logs. It is designed with only MTA log files in mind, and only those analysis scenarios that would apply to the data contained within them. Should a web application resident on a system utilize an underlying mail service, the correlation of events within the web service and the email logs would be of great importance. Investigations of computer related incidents must provide evidence that places the suspect with access to and control of a system during the time of the incident [43]. The use of email correspondence may be a vital avenue of communication between malicious parties, and includes timestamped headers that record each message that is sent. The ability to produce a timeline of events prior to an investigation yields better evidentiary representation for an investigation.

3.1.4 AWStats

AWStats [114] is a Perl-based open source reporting and statistical tool that is able to analyze and detect web robots and a variety of worms [326, 114]. As a web application, the internal methods of AWStats allow to check for XSS attacks against itself [113]. It is puzzling that this functionality is provided for the tool itself, yet not built-in to detect XSS attacks for the service being monitored, nor within the services logs being analyzed. As long as the administrator has setup the extended log format for the web service or in the case of other services setup the log to include the necessary information, AWStats can provide this information in the format of a detailed report. Examples of analysis data include domains or countries of visitors, most viewed pages, authenticated visits, the detection of robots, operating system detection, referrer search engine detection, and visitor browser detection [113]. AWStats does not only analyze the AWS log files in the CLF, and ELF, but can also analyze MS IIS W3C, other File Transfer Protocol (FTP) [314], mail server, proxy, streaming media, and a multitude of web server log files [113]. Technically, it is not even limited to these predefined log formats at all, as the end user can define their own log formats for analysis. However, the information gathered is still limited to the data contained within the logs themselves. As was stated in the examples above, AWStats has many predefined reports. It also includes the capability to define user reports. In addition to the reports options and filters remaining completely reconfigurable, the code is also distributed in a modular fashion, thus enabling additional features to be implemented by way of AWStats plugins [115]. Reports can be generated statically or dynamically from the command line or CGI interface and can be output in HTML, XHTML, Portable Document Format (PDF) [7], or sent to standard output to be exported. This tool was elegantly designed for the purpose of the analysis of web related log files. It was not designed as an overall log file correlation engine or forensic tool. The data collected within the reports of this tool can greatly assist during an investigation. With additional correlation capabilities, support for other log files and process monitoring would provide the characteristics required to perform forensic log analysis. The extensive statistical reporting generated by this tool to provide additional web analytics, which can be correlated to events within the forensic timeline. The extensibility of this tool greatly increases adoption capabilities by forensic investigators.

3.1.5 Breadboard BI Web Analytics

As a plugin, the Breadboard BI Web Analytics [250] module provides yet another statistical analysis view as a supplement to an overall business analytic open source framework called Pentaho [250]. It performs summary statistical analysis from web log information after it integrates web logs into an OnLine Analytical Processing (OLAP) [125] cube model and presents an overview that provides a comprehensive business perspective. Talend [218], a commercial product also produces an open source package called Extraction, Transformation and Loading (ETL) [365] that is used to integrate data into a cube from any source [218]. The aggregation of data from multiple sources utilizes several standards for different interfaces. This is a definite advantage when used in an application for the correlation of data sources. Also known as web analytics, the measurement, collection, analysis, and reporting of internet data for the purposes of understanding and optimizing web usage can be effectively utilized for marketing purposes in a business setting [19]. Pentaho integrates three report views for Breadboard BI, described as generalized, analyzed and dashboard. Although web analytics may give an excellent overview, it will not generate enough evidentiary data, or provide sufficient results to use the reports in an investigation. This tool would form an excellent basis for a log analysis tool, as when dealing with a large amount of data from different sources, the OLAP cube would enable correlation, a method of organization, and storage of data which would permit efficient search capabilities.

3.1.6 Calamaris

Written in Perl 5, `calamaris` [32] is an open source program used to parse and generate ordinary or framed HTML output that includes text and graphical reports on proxy service logs. It is limited to log files from the proxy service packages Cisco [212] Content Engines [211], Compaq Tasksmart [200], Inktomi Traffic Server [147], Netscape also known as Iplanet [295], Novell Internet Caching System [66], Web Oops! proxy server [351], Squid [227] or related proxy log files [32, 311]. The analysis included in the available reports summarizes usage, freshness, request methods, incoming UDP and TCP requests, outgoing requests, neighbour caches, top, second and third level domains, protocols utilized, content types and extensions, host statistics, time ranges, and error codes [32]. `calamaris` is run from the command line, however not all of its options can be implemented as such, and directives must be placed in a configuration file. As a summary statistical tool, this excels at what it was engineered to do, as part of an overall analysis system, any network utilizing a proxy service that needs to be monitored would benefit from the analysis provided by this tool. The log analysis tool `calamari` has very specfic target parameters and would function only as part of an encompassing log analysis engine.

3.1.7 Chklogs

As an open source log management tool, `chklogs` [189] performs compression, rotation, log grouping with pre or post processing, and threshold or action definitions for each log file [189]. It is written in Perl and associated with a separately designed user interface. It is also released with methods to define extensions to itself, and it includes a user resource configuration file. Any manipulation performed on the log files, should they change the log files in any way, would have a negative impact, if provided as evidence. The chain of custody must be upheld and any manipulation performed on that data for the purposes of finding the information currently must be performed on a digital copy of the evidence. Presenting the raw data as a point to formulate a conclusion, might not easily be understood by legal representatives. Only unmodified data can be used as evidence [339], therefore it is the investigators duty to accurately represent and provide meaning to the data. Should log files be stored compressed, there should be a mechanism of appropriate decompression within a log analysis tool. The extraction of pertinent data, an indication of the original location of that data, and the ability of an investigator to provide a description for what a particular event means, would be advantageous components within a forensic log analysis tool.

3.1.8 CORE Wisdom

Aside from providing integrity for and centrally managing log files, CORE Wisdom's [116] greatest contribution is the ability to process and generate unique graphical representations of the log data beyond mere pie charts and rudimentary graphs [116]. As such, this is done to alleviate the problem of missing important events within log files, and draw attention to or extract necessary patterns or trends of events that should cause alarm more easily for the analyst. This is a proprietary program that runs only on a MS Windows environment, however it can read from any type of log file [367]. The problem with this program, as with many log analysis systems, is that the analyst has to define the rules for importing the log file itself. The number of log file types that need to be defined correspond to the complexity of the target network setup and the logs that need to be analysed. This tool does not provide any correlation ability. The analysis this tool performs is accomplished in real-time. If the analyst does not know what visual cues to look for or cannot define events that should be flagged as alarms, then representing it in a different format does not provide a more effective analysis methodology. The visual representations offered by this tool may provide the necessary understanding to enhance evidentiary reports prepared for the analysis of log data.

3.1.9 EventLog Analyzer

Implemented as a web application, ManageEngine's EventLog Analyzer [262] tool provides centralized management, real-time analysis and reporting of a variety of log file types. Compatible log file types this tool can provide analysis for include the MS Windows Event log [268], IIS W3C FTP and HTTP server logs, MS Structured Query Language (SQL) [96] server [86] logs, Dynamic Host Configuration Protocol (DHCP) [122] logs for Windows [75] or Linux [288], International Business Machines (IBM) [72] AS 400 logs [67], syslog [179], or Oracle 10 GR2 audit logs [291] [262]. This tool is also capable of generating archive files, which can be stored for later analysis. Centralized management functionality ensures network wide application, security, system or syslog device logs are collected, normalized, and archived for analysis into a built-in, central MySQL database [262]. EventLog Analyzer can define automatic alerts, generate historical trends based on system events, group host information together to show interactions, show failed logins, malicious users and show applications that are causing performance or security issues. EventLog Analyzer also includes pre-built reports and the ability to choose the data and format for the generation of custom reports and templates. Reports are output at specific intervals and can be exported to HTML, PDF and Comma

Separated Values (CSV) [344] formats. The analysis can generate both graphs and text-based representations as output. The tool retains eleven groups of default report styles as well as any number of combinations for custom generated reports [262]. Derived from this solution, both the archival of logs and several default reports ensure compliance with the Federal Information Security Management Act (FISMA)[1] [342], the Gramm-Leach-Bliley Act (GLBA)[2] [62], the Health Insurance Portability and Accountability Act (HIPAA)[3] [63], the Payment Card Industry - Data Security Standards (PCI-DSS) [306], and the Sarbanes-Oxley Act (SOX)[4] [229, 262]. The principles of forensic process in different scenarios vary widely. Understanding and implementing compliance with laws in which the forensic investigator operates would enable an analysis tool to garner widespread adoption for that geographic area. This log analysis tool performs many qualities essential to a forensic log analysis tool, but does not include automatic correlation between log files, or extensibility with regards to the types of log files that may be analysed.

3.1.10 Ftpweblog and Wwwstat

`ftpweblog` [350] was built to extend the capabilities of a program known as `wwwstat` [350]. The program `wwwstat` is used to output website access statistics information to an HTML file. However, this program only comes with a default report style, and the methods of generating additional information or manipulating the reports could only be achieved with great difficulty. When `wwwstat` is integrated with `gwstat` [242], this output can generate graphical representations of `access_log` data. The `ftpweblog` was designed to integrate `wwwstat` and `gwstat` not only to different log file formats, both HTTP and FTP, but also to extend the configurability for the generation of reports [132]. Both `ftpweblog` and `wwwstat` are Perl based programs that utilize regular expressions. They are published as free software. Neither tool provides the necessary functions to operate as a complete log analysis solution.

3.1.11 Funnel Web® Analyzer

The freeware multi-platform[5] web analysis tool Funnel Web® Analyzer [216], utilizes information from W3C formatted web server access log files to generate reports that document both internal and

[1]There is no Canadian equivalent for this Act. The government of Canada, United States Government Procurement website refers back to the American act [284].

[2]The Canadian equivalent for GLBA is Bill 362 or Personal Information Protection and Electronic Documents Act (PIPEDA) [303].

[3]In Canadian law, contents of this act is also covered in PIPEDA.

[4]The Canadian equivalent of this is Bill C198. [345]

[5]MS Windows, Linux, Solaris [293], and Mac OS X [69]

external server statistics. It includes fifty default customizable reports that contain both graphs and textual data that can be published to PDF, HTML, MS Word [83], Rich Text Format (RTF) [73] or MS Excel [82] file types. The enterprise version of Funnel Web® Analyzer, enables additional features such as real-time streaming analysis, extended feature changes for reports, an innovative click stream report per visitor, advertising statistics, diagnostic reports, conversion of log files to W3C standards, support to analyze larger files with a flush mode and up to 1000 virtual domains [216]. Although this program does not use a database, it configures files such that they become easily imported to one. The tool Funnel Web® Analyzer also performs no correlation and is limited only to the analysis of web log files.

3.1.12 Http-analyze

`http-analyze` [324] is a fast multiplatform web server log file analyzer that can process data in only three log file formats, the CLF, ELF and Distilled Log Format (DLF) [378, 324]. The analysis performed by this tool includes automatic log file type detection. The `http-analyze` tool includes the option to genenerate one of two different standardized HTML reports to include statistical and access load information summaries. These reports include graphs, tabulated data, and three dimensional forms [357]. Real-time analysis is only acheived by scripting the rotation of the log files, used in conjunction with the automated calling of this tool. The `http-analyze` tool does not generate or format the log file information in any way, (other than reading from compressed files where it performs decompression [357]). It does not put the information into a database, nor does it perform correlation between the web server files with any other system available information. There exist many freely available tools that can convert other non-standard web log formats to one of the three types that `http-analyze` can analyze.

3.1.13 Logjam

Logjam [64] is a web application whose functionality implements a web traffic analyser that provides in-depth statistical analysis of W3C ELF log files. It has prerequisites of a MS Windows server installation that includes Active Server Pages (ASP) and Microsoft Data Access Components (MDAC) [80, 64]. Preliminary to analysis, log files are jammed into an SQL database for analysis [64]. Default reports and a customizable report generator that creates SQL queries based on user preferences are included with this tool [65]. This application is limited to the web server log files produced on a MS Windows machine, and runs only on a MS Windows machine. It has neither implemented correlation nor does it function in real-time. Logjam does have the advantage that it generates specific website data such as a user clickstream analysis. Reviewed in the reports, this advantage is very useful in determining a forensic account of user interaction.

3.1.14 Logparser

MS has produced an SQL-like query adapter for many auto-detected log file types including System, Security and Application Windows based log formats. Logparser is capable of analyzing all IIS log file formats, including IIS logs with W3C fields, text file based formats, Domain Name Service (DNS) [271] logs, generic NCSA log files, HTTP `Error_log` files, log files adhereing to W3C standards, as well as CSV, Tab Separated Values (TSV) [241] and XML based file formats. Logparser is capable of parsing specialized binary files (such as those from MS Network Monitor [85] and ETW Trace [78] files), retrieving system information (MS Event Log, files and directories, registry keys, and Active Directory Object (ADO)s [13]), and processing custom data through the use of user implemented input format plug-ins [183]. As a toolkit, Logparser does not provide a graphical user interface, but provides functionality through a command line invocation by script (for use with external files), or direct manipulation of queries by prompt interface. There exist two programs that provide this convenience listed as Logparser Lizard [6] and Visual Logparser [7]. The Logparser language includes more than 80 functions in different categories with examples that perform string manipulation, arithmetic operations, and provide access to system details. Each of these functions can modify or manipulate the content of fields in some manner. The log file conversion capabilities of Logparser aid in the process of adapting log files to queries for performing analysis including correlation. For correlation, Logparser has the capability of combining the data from multiple sources, and then performs queries upon it. Logparser produces output in standardized formats such as CSV, TSV, XML, Syslog, W3C, IIS, SQL, and non-standard formats, which require either immediate presentation or include graphical output such as DATAGRID [182], CHART [182] and NAT [182]. Limitations as documented from one particular website [23] of this tool include the requirement of running MS Office to generate graphical output, and the automatic detection for CSV files default to string with no method of defining the columns manually. Logparser does not include methods of analysis, only the strength to perform the queries. The user must create useful

[6] www.lizardl.com
[7] http://www.codeplex.com/visuallogparser

queries to satisfy any analysis requirements. There are many places of reference that will describe and implement methods employed to analyse situational instances. For example, Logparser has been used to monitor user activities, monitor system file integrity, check for SQL injection attacks, check for excessive failed logon attempts, determining malicious modification, identification of brute force attacks, and reconstructing intrusions.

3.1.15 Lire

Compared to all other open source log analysis software, Lire [152] can provide analysis for the widest variety of log file types through the use of conversion tools that convert the log files to DLF. These tools are provided with a default installation of Lire, as are thirteen default reports that can be modified and customized. Available default report templates include a statistical analysis of database, dial-up, DNS, DNS-zone[271], email, firewall, FTP, print, proxy, Spam filter, Syslog, and web server log file groups, which are generated through the use of an XML-driven reporting engine [152, 153]. This tool does not provide correlation among log files, and its interface functions as a log viewer or manager that configures timely reports on previously recorded logs in batches. Batch generation does not translate to real-time performance as log files must be closed, and not open for writing at the time of their conversion and the instantiation of analysis functions.

3.1.16 Logrep

The collection and presentation of data contained within log files of over 25 reporting tools including Snort [263], Squid [227], Postfix [379], the AWS, Sendmail [343], Iptables [26] or Ipchains [336], Syslog, Xferlog [170], Firewall-1 [366], Wtmp [175], Oracle Listener [100], Pix [364], and MS Windows NT event logs is achieved by the tool `logrep` [235, 178]. This information is deliverable via Secure SHell (SSH) [393], or over HTTP as formatted HTML reports that include multi-dimensional analysis and graphs. This tool does not provide any correlation capabililities.

3.1.17 Logstalgia

This innovative website traffic visualization analyser presents the AWS `Access_log` entries by listing event requesters on one side of a pong battle board and the pages requested from the web server on the other [44]. Colored balls represent the requests being made, the same color as the host making the request and they travel across the screen to hit the requested locations [44]. Failed requests pass by the paddle on the server side, and successful ones are hit by the paddle [44]. To view individual requests, the visualizer may be paused, and on mouse hover over the request to see the actual data. This is severely limited as a log analysis tool in that it is only used for visualization. However, it is both entertaining, as well as useful in showing a new method of visualization for an investigator. The tool can run in real-time or utilize a log file as input.

3.1.18 Mywebalizer

Highly detailed and easily configured HTML reports covering web server usage statistics can be generated during the execution of the open source project called Mywebalizer. The code is written entirely in the C programming language which enables it to be extremely portable to operating systems conforming to POSIX standards (Linux, Unix and Solaris). The log file formats that it can analyze include CLF server logs, several variations of the NCSA Combined logfile format, wu-ftpd/proftpd xferlog FTP format logs, Squid proxy server native format, and W3C Extended log formats [29]. Log files that include Internet Protocol Version 4 (IPV4) [316] and Internet Protocol Version 6 (IPV6) [112], DNS lookup and geolocation services are supported or provided as built in functionalities [29]. In addition, this tool provides decompression capabilities so that bzip2 and gzip compressed logs may be used directly without the need for uncompressing [29] which saves on space, and provides analysis compatibility for larger log files. The analysis performed by this tool summarizes statistical information about the web server as a whole by sites or time based report styles in both tabular and graphical formats that are highly configurable from a command line invocation [30]. This tool does not provide automatic detection of log file type, nor does it run in real-time or provide any correlation capabilities. With these functionalities in mind, it could not provide the necessary characteristics of an encompassing forensic analysis tool.

3.1.19 Open Web Analytics

Written as a web application using PHP, the Open Web Analytics [5] platform is a generic web analytics framework that can provide analytical data regarding any web application to existing sites and other web applications [5]. Due to its nature as a web application it can function on any operating system that contains a browser and it can easily be added to existing popular web applications using JS, PHP or REST application programming interfaces. Open Web Analytics provides built in support for Wordpress [123] or MediaWiki [54] applications [6]. The Open Web

Analytic tool's main function is to provide real-time tracking, monitoring and reporting of web usage statistics. Some examples of information it can provide include visitor click streams, geolocation of visitors, Really Simple Syndication (RSS) [387] subscriptions made to the site, browser information and web application specific features, such as indentify Wordpress visitors by username or MediaWiki visitors by article or email address [4]. It also provides the means to develop additional functionality through the use of plugins.

3.1.20 Pyflag

Pyflag [52] is an open source, web based application that performs forensic log analysis through an extensive Graphical User Interface (GUI). This tool is capable of handling large volumes of log files in many different file formats, disks or images, and network traffic data such as `tcpdump` data [52]. Data can be added to a MySQL database for faster queries but the log types are still specified by the end user and due to the fact that this tool's basic log function is only to view the log files, the analyst must perform the required analysis using prior knowledge and experience. No specific regular expressions that could be used to find attack scenarios are listed within the documentation for this project, so although this tool allows for regular expression entry and their operation on the data, there exists no prebuilt analysis capabilities in this tool. The interface for analysis of log data includes querying, sorting and graphic representation of the log data. Similar to a statistical analysis log analysis functionality, but with the option to analyse many formats other than just log files pertaining to web applications.

3.1.21 Sawmill

The major advantage that the Sawmill [135] architecture features is that it includes plugins for and automatically detects over 800 distinct log file types,[8] and provides a method for defining plugins for non standard log file types. The architecture includes the components log importer, the Sawmill database, a reporting interface, a web server, a command line interface, a scheduler, and two languages used to manipulate data and how data is stored for analysis. The Sawmill language or Salang is used to display pages and to define log filters. It is structurally similar to Perl or C, and contains the elements necessary for the definition of filters including regular expressions, and conditional logic [135]. The other language included within the Sawmill architecture is the Sawmill

[8]A comprehensive list of the file type definitions predefined are listed here: http://www.sawmill.net/cgi-bin/sawmill7/docs/sawmill.cgi?dp+docs.technical_manual.logformats+webvars.username+samples+webvars.password+sawmill (Note: this data pertains to Version 7, meanwhile, the current Version is 8)

structured query language. This language is a subset of most SQL commands, and is utilized to access internal database information within predefined table sets of the Sawmill database [134]. The separate parts that comprise the Sawmill architecture all perform functions essential to an efficent log analysis tool. To provide a forensically sound basis, external methods to ensure the original log data has not been tampered with, would need to be instantiated. This tool provides the access necessary to perform correlation between log data sources, but does not perform this function inherently.

3.1.22 Squidj and Scansquidlog

Squid [227] is a web proxy cache service that is used to store frequently accessed pages, which lowers bandwidth usage, enables more efficent access to stored sites, and can optimize network traffic by mapping server hierarchies then appropriately routing content requests [227]. When utilized in a network, all requests are routed through the proxy service to achieve the above mentioned benefits, and events are logged. The two main log analysis tools utilised which apply specifically to Squid are Squidj [282] and Scansquidlog [33]. These two tools are discussed here because the ability to correlate their results with data from other system logs such as those generated from the AWS, other varieties of web servers, or applications could provide convincing corraborating evidence. Squidj is a refresh pattern analysis tool written in Python that gives an overview of all the objects in the cache and how they are accessed. It utilizes the two modules[9]: `weblog` which enables web log file parsing, and `conf` which parses configuration files [282]. One forensic example would be to compare the original bytes sent from a stale file to the byte count to the number of bytes recorded in the access log event for that page, or a large change in the number of bytes in a new version of a posted website file. Scansquidlog is utilized to search for specific URL string matches within the squid log [33]. It not only finds the URL specified, but also lists all branches beneath the specified URL. This method of extracting the strings is much faster than searching using regular expressions. However, the method's disadvantage is that it requires the investigator to individually specify the URLs that they are looking for. Given that these two tools are written to specifically analyse squid logs, they are insufficient for a forensic log analysis tool on their own. They do include statistical analysis, which when used in conjunction with other data might yield interesting results.

[9]These can both be found at http://www.pobox.com/~mnot/

3.1.23 Swatch, Logsurfer and Tenshi

The reason for these three tools to be grouped together, is due to the fact they provide implementations that are driven through the use of regular expressions. Each employs a different methodology, analysis and provides their own output.

Swatch [21] is a Perl-based tool originally designed to act as a real-time monitoring process for Syslog messages. It has been extended to monitor any type of log format [21]. Regarding functionality, what Tripwire [224] and Advanced Intrusion Detection Environment (AIDE) [251] do for file integrity monitoring, Swatch does for log file monitoring [20]. Analysis of the messages entails that those events being sent to Syslog are attempted to be matched against rules defined by the user. Should a match occur, an alert or other action then takes place. Configuration for Swatch is user-defined in a file called `.swatchrc` that includes regular expression patterns followed by specified actions to take place when the patterns are successfully matched. Two examples as specified in [31] include a method of detection for URL buffer overflow attacks and a method to detect DoS attacks. When an attacker attempts to perform a buffer overflow attack through the request of an overtly long filename, the AWS logging service will respond and log a "File name too long" event. The corresponding Swatch configuration entry to detect this phenomenon would appear as follows:

```
watchfor /File name too long/
  mail addresses=gc\@majortom.cx,subject=BufferOverflow_Attempt
```

One method that detects a DoS attack initalized by Swatch, would be to define a throttle HH:MM:SS action after with the `watchfor` regular expression. This example is demonstrated as follows:

```
watchfor/http:\/\/www.youtube.com\/watch\?v\=dQw4w9WgXcQ/
  throttle 00:00:30
```

This example watches for a specific URL that has been submitted by a user, and gives a threshold of 30 seconds, should someone repeatedly attempt to submit that same URL as part of a DoS attack. Swatch is an excellent monitoring tool, however, when determining report styles, the log of the swatch service does not resemble a report. It does not offer statistical analysis, unless the analyser takes the data from the monitoring session, and performs analysis on it separately.

The Tenshi[10] [355] tool (formerly known as Wasabi) is a Perl-based log monitor that reports on regular expression matches of events in one or more log files [355]. Tenshi is a complete rewrite of the C program Oak[11]. The regular expressions defined by Tenshi are assigned to a queue containing user-specified actions to be performed within an alert interval [355]. The actions that can be taken may include notification emails or periodic reports. These reports include selected matched event information. Redundant or non-essential event information can be excluded through the use of regular expressions in the reporting style definition located in the Tenshi configuration file. The report that is generated by this tool is output as a CSV file that includes the hostname the log file pertains to, the log itself, and the total number of hits or events [356]. As a forensic tool, Tenshi provides real-time monitoring and time-based reporting. However, the performed analysis is limited to regular expressions defined by the user and there are no default pre-built expressions.

Extending the functionality of Swatch, Logsurfer [369] performs real-time log file monitoring and reporting. One example of this extended functionality includes the ability to group related log events and output them to individual reports. This tool being written in the C language enables the capability to monitor large volumes of log data [369]. The Logsurfer tool is highly adaptable in that it can be configured to monitor any type of log file for port scans or the disappearance of log event entries. One method of detecting port scans would be a generated alert when a large volume of dropped packets is being logged from a single IP source address. Rules and contexts are the two functions utilized to perform Logsurfer log processing. The contents of each Logsurfer rule is structured as follows:

```
match_regex not_match_regex stop_regex
not_stop_regex timeout [continue] action
```

The `match_regex` field defines lines that will match the rule. The `not_match_regex` field is used to define those lines that do not match the rule. Interestingly enough, rules can be predefined to remove themselves from the active rule set using the `stop_regex` field and exclusions to removing the rule defined in field `not_stop_regex`. To define a time limit or interval for which this rule should remain active for, a `timeout` value is specified. If no limit should be placed a `timeout` value of 0 should be defined. To override the default behavior of Logsurfer that asserts line processing is complete when a successful match is made, the `continue` keyword will be specified. This ensures

[10]The Tenshi tool is currently maintained at www.inversepath.com.
[11]The Oak program is included in the package of tools at www.ktools.org.

that Logsurfer continues to process lines that would match the rule. Unused fields employ `-` as a placeholder. Finally, a rule `action` is specified and can be `ignore` which ignores the line matched, `exec` which executes a program specified, `pipe` which sends the log event to the command line, `open` which defines a new context, `delete` which deletes an open context, `report` which executes a program, piping to it the context data, and `rule` which creates a new rule. Contexts enable grouping functionality even with messages that are interspersed with non-related messages as mentioned earlier [370]. Logsurfer is released in two different formats for Unix and Solaris-based systems. There exists extensive documentation for Logsurfer that provides many log analysis tips such as being cautious when processing the log files, as control characters may be maliciously inserted. As with the other tools within this category, this tool requires an external program for the generation of specific reports. However, being able to group log entries in real-time can have a significant advantage over the other tools in this category. Information included with Logsurfer does not provide predefined regular expressions. These must be obtained from other sources such as [12] [13], or through user definition.

3.1.24 Visitors

Providing unique web log statistical analysis based on access request events, the visitors [337] tool takes web logs as input and returns either an HTML formatted report readable by any browser or textual data that can be piped to any source or remotely retrieved over SSH [337]. Written in the C langauge, it is highly portable and capable of handling large file sizes. Analysis for this program can be extended through the use of regular expressions, and the ability to pass data to the Graphviz [126] program for the generation of visual representations of statistical information. Extracting data from Google [14] search URLs can also provide interesting keyword patterns to match. Adding this analysis to a user trail's data and correlating with other data sources could provide corroboration. As a freely distributed program it includes a list of analysis techniques provided by this tool. Explanations and examples are provided within the manual page included with the source code[15]. Combining analysis data and techniques from several different web log analysis tools such as visitors may enable a better user web trail to emerge. Thus, as a forensic log analysis tool, it provides very limited functionality. The combination of analysis methods provided could enhance an encompassing product.

[12]EMF's Logsurfer configuration page describes useful Logsurfer configurations: http://www.obfuscation.org/emf/logsurfer.html

[13]CERT-dfn official Logsurfer site: http://www.cert.dfn.de/eng/logsurf/

[14]www.google.com

[15]The source code can be located within the current Debian repositories, or at the authors website: http://www.hping.org/visitors

3.1.25 Weblogmon

Written entirely in Perl, Weblogmon [245] monitors server log files and reports usage information regarding current users. Identification of users is based on their IP address, their HTTP logged email username values determined by `ident` values, or cookie information [245]. It can resolve IP addresses to domains utilizing the `perldns` module [246], and can understand the CLF, the ELF, and the ELF which appends the `mod_usertrack` cookie information [244]. From these formats, it can auto detect the type the user provided as input, and it adjusts the output report to include pertinent analysis automatically. This tool is limited to those three types of log files, and does not perform any correlation. Relative to other log analysis tools, Weblogmon provides limited documentation and functionality. It was included in this study as it provides a unique perspective on how to track or monitor users.

3.2 Log Analysis Development Frameworks

The difficulty in providing an encompassing, effective and efficient forensics tool lies partly in the size of the event log, but other factors also include the simplification of those event logs, and appropriate functionality by network configuration requirements. For example, a most frequent false positive may generate an alert via the detection of an attack event, but this attack can only be successful should a specific application be running on an intended target. The ability to specify specific rule sets, signatures, and the configuration of the network becomes an imperative, yet daunting task to the investigator. With the following log analysis development frameworks, an investigator or a target company may be able to fine tune, or develop an application specifically for the purpose of an investigation prior to or after an incident has occurred.

3.2.1 `Apachedb`, `mod_log_sql` and the Apache::DB Project

The use of databases in the realm of the log file analysis is an essential design decision, which tends to speed up performance, alleviate issues regarding organization as well as enable prospective correlation for any analytical tool. `Apachedb` [215] is a small PHP script utility that transforms log data from one of four formats, namely `common`, `combined`, `mod_user_track` or `squid`, into a Mysql^{TM} database for further analysis [185]. It provides functionality to import a pre-generated log file into the database, and a method to write directly to the database while the web service is in operation [215].

Since the inception of the Apache::DB project and `mod_log_sql` [231], the `apachedb` PHP script has lost its usefulness as these two software packages offer the same functionality of configuring the AWS to log directly to a database. Functional packages related to logging to a database provided by the Apache::DB project include Apache Derby [318], Apache Torque [144], Apache Java Data Object (JDO) [143], Apache Object Relational Bridge (OJB) [139] and Apache DdlUtils [140] [142]. These Apache packages provide connectivity to the associated technologies mentioned in the respective package names. Derby is an open source database [318]. As an XML-based object relational mapper [144], Torque serves as a connection manager to a variety of databases. JDOs are useful when accessing or manipulating persistent data contained within databases [143]. An OJB exists as an object or relational mapping tool that allows transparent persistence for JDOs against relational databases [139]. The Apache DdlUtils package works with Database Definition File (DDL)s [140], which are XML files that contain the definition of a database schema [140]. As with most log analysis tools that have been reviewed at this time, these tools are development components, some of which are not entirely implemented. For their effective utilization for the purposes of rendering forensic log analysis, they must be employed by a developer or investigator as a part of a personalized toolset.

3.2.2 Cascade Software Packages

The Cascade Software packages [213], developed by Cogent Real Time Systems include the Cascade Data-Hub, the OPC Data-Hub, Historian and TextLogger. These four programs could provide an Application Programming Interface (API) to be utilized for the development of a very effective log analysis system, IDS, or provide real-time intrusion protection services. The Cascade Data-Hub is a real-time data collection and bridging hub for modular Windows applications, and normal or embedded Unix applications [213, 50]. It is entirely resident in volatile memory and allows a developer to share data among any number of programs over TCP or Dynamic Data Exchange (DDE) [77] connections [50]. This application enables modules to communicate without deadlocks through the combination of data packaging and intelligent queueing, which ensure that the behaviour and communication of one program will not adversely affect any others. [50] The API for Cascade DataHub consists of a C code library that allows for reading, writing, and program registration to generate exceptions with the DataHub [50]. Object Linking and Embedding for Process Control (OPC) [149] is a series of standards specifications regarding the communication between Windows programs and industrial automation systems, which operates using a client-server model. The OPC

Data-Hub is similar to the Cascade Data-Hub in that it operates as a data collection repository and bridging hub, yet it connects all operating systems, as well as OPC servicable devices. As an event-driven data storage management program that maintains persistent time-sequence data sets derived from process data, the Historian also offers a query facility suitable for generating export data or graphical representations [51]. The four built-in queries within Historian include a raw data view, periodic data view, relative interpolation, and periodic relative interpolation. The program also provides a method to develop additional request types. Textlogger performs real-time collection of process data and generates log events within regular ASCII files. This last tool would enable a developer to generate log data for processes that does not perform logging inherently. A log analysis system built with this as a base would be completely extensible, able to conform to individual needs of certain industrial setups, and would remain adaptable for correlation, reports, and queries.

3.2.3 Crystal Reports

Crystal Reports [380] is a Windows-based commercial software distributed by SAP[16] that provides the ability for a developer to define and perform desired analysis on multiple sources of data. The information that are used in these customized reports can come from a variety of sources, such as an OLAP cube, MySQL database, or flat files. This software provides a platform to perform business analytics. It has the capability to be extended towards web server analytics, which is referred to as web intelligence, but all analysis performed with this in mind must be uniquely specified. Crystal Reports enables the development of such applications through the integration of this application with .NET [76], Java, or Java 2 Enterprise Edition (J2EE) [281] [380]. Internal log analysis functionality does not exist, as this tool's primary function is as a reporting tool. It can be programmed to achieve overall business analytical tools, or perform log analysis specific functions depending on the application developed that may utilize these reports as an output format. Crystal Reports was engineered to function as a development tool. It must have plugins or individual reports developed around it to become functional as a log analysis tool, or even a business analytics tool. Default report structures are available to download from the product website, but may not include all functionalities needed for processing log files. Capabilities other than report layout, summary statistics or data formats would need to be programmed by the application outside of the Crystal Reports API.

[16]http://www.sap.com

3.2.4 JasperReports

Available as an open source tool suite, JasperReports [60] is packaged within a collection of business intelligence suite of tools that include Ireport, Jasperserver, Jasperanalysis, JasperETL, and Jaspersoft. This is a reporting tool not specifically designed for log analysis. Much similar in functionality to the commercial Crystal Reports software, a developer can specify within JasperReports how they wish to accomplish log analysis and formulate reports based on data source connections to a Java application. JasperReports can be connected to the development environment Eclipse [165] and integrates with the Pentaho [60] suite for business intelligence [60]. Report templates are generated through XML files, and can be created with the aid of Ireport. Ireport is the visual designer that generates the reports. Ireport includes builtin query support for Enterprise JavaBeans Query Language (EJBQL) [269] from J2EE, Hibernate Query Language (HQL) [323], Multi-Dimensional eXpressions (MDX) [87], SQL, XPath [47], in addition to custom languages such as Procedural Language / Structured Query Language (PL/SQL) [292] [226]. Reports can be exported in many different formats including CSV, Excel, HTML, OpenOffice Data Format (ODF) [294], PDF, text, RTF and XML [226]. Correlation may be added but all analysis is customized and developed by the analyzer.

3.2.5 Simple Event Correlator (SEC)

The Simple Event Correlator (SEC) [376] is an open source, platform-independent event correlation tool that can operate in an offline or an online (real-time) mode. Within this context, event correlation has occured when a conceptual interpretation of multiple events has generated a new meaning outside the context of each individual event [225]. A discussion of the benefits of correlation was outlined earlier in this chapter and previously in Chapter 2, and possible future applications regarding correlation is given within the Conclusion. This development tool has previously been integrated into applications that provide network management, system monitoring, intrusion detection, and those that perform log file monitoring with analysis. Methods included in this tool define the event correlation rule types. Each rule includes an event match condition, an action list and an optional boolean expression that determines whether the rule should be applied at the specific time interval it was matched in. The event match condition is defined by a regular expression, boolean expression, perl subroutine or by direct substring match [374]. The different rule types defined by this tool include single, single with script, single with suppress, single with threshold, single with two thresholds, pair, pair with window, suppress, calendar, jump and options [376]. In the following, we provide an explanation exerpted from the manual [376] for these rule directives in order to demonstrate the functionalities of SEC. The single rule matches a specific event and then executes an action list. The single with script matches an event, then the execution of the action list is dependent on the exit value of an external script or program. Single with suppress matches an event, then ignores subsquent matching events for a specified duration. A single with threshold rule is used to generate a match if a given number of events are matched within a specified duration, any additional matches found during the duration window are ignored and the calculation of future matches are made against sliding window durations. When two thresholds are given for a single event, the first threshold specifies the count to successfully match, this activates the first action list. The second threshold is for the continued match, ensuring that the number of matched events remains above the number specified in the second threshold, which activates a secondary action list. For a pair rule, the first event is matched and an action list is then executed, any further events matched are ignored until the second event specified is matched, and then another action list is invoked. The pair with window rule matches an event and waits for another event to arrive within a specified duration. Two separate action lists are given that specify the boolean outcomes determined by the success or failure of finding the second event. A suppress directive indicates an event should not be matched by future rules. Calendar directives are used to execute action lists at specific times. When a jump directive is specified within a rule, it submits those events matched by the rule to other rule sets for further processing. Any extraneous variables to be included in a rule set are specified using the options directive. Subsequent actions may include the creation or deletion of contexts, the execution of external shell commands or external programs, and the generation of new events. Other available actions that may be performed are specified within the manual page [376]. This tool includes a large online community that provides application specific rules [375] for different network environments. As another development tool, this performs only one function that, in our opinion is essential to an effective forensic tool. Extensive configuration is still required for this tool to operate for different logging environments.

3.3 Intrusion Detection and Integrity Monitoring Tools

Although these two classification of tools do not necessarily base their results upon analysis of log files, they provide insight into methods, which can be utilized in a forensic analysis tool. An IDS is designed to generate alerts based on the presense of malicious actions designed to intrude a system. An enormous amount of research effort over an extended duration has been devoted to the development of state of the art techniques to be employed by intrusion detection, or Intrusion Protection System (IPS)[17]. It is logical to leverage the capabilities of these methods when defining attack rules or signatures, enabling the reporting of IDS to be utilized as a source of cross-correlation. Another aspect that is critical to a forensics prospective is shown when the capture of evidence provided by log files becomes inadmissible due to a lack of integrity. An implementation of components that can ensure the efficacy, admissibility, corroboration and transparency of evidence require the introduction of aspects previously implemented by or those that can be verified using integrity monitoring, IDSs, and IPSs.

3.3.1 ArcSight Logger and ArcSight ESM

Management of logs requires the ability to parse, normalize, organize, search, generate reports from and store all of the data in a useful scalable manner. ArcSight Logger [210] includes a useful analysis portal that allows access to dashboards, the generation of pre-built and user-defined reports, the ability to perform efficient searches utilizing regular expressions, simple strings or predefined field values [208], and settings for real-time alerts, or event forwarding. Prebuilt reports come with scheduling interfaces, access controls, a variety of export formats, and other flexible features [346]. Arcsight utilizes "connectors" to provide the capability to collect, categorize, and normalize over 300 distinct log types parsed into the Common Event [256] format [210]. Further extensibility is provided by the use of ArcSight's flexconnectors, which can be used to collect custom or in-house data sources [210]. ArcSight Logger is made available in the form of an integrated system appliance, or is distributed as a standalone software. The logger tool only provides the capability to effectively manage the logs. More analysis is performed by correlating the information organized by Arcsight's Enterprise Security Management (ESM), as the logger can forward security events to this tool, as well this tool gathers data from a variety of different sources such as network devices and connected workstations. Due to the proprietary nature of both ESM and the Logger, it has been difficult to find appropriate documentation to determine if the software includes pre-defined attack definitions and their method of correlating these facts to other source events as necessary to confirm attack detection. It has been claimed that this correlation to provide evidentiary information can be accomplished by

[17]See http://www.snort.org for the most widely used open implementation of an IDS or IPS and [259] for an example which overviews methods in intrusion detection research dating before 1993

this leading industry tool [210, 209].

3.3.2 Guard26

The Guard26 [361] implementation is a real-time log parser that represents a first attempt at a Linux IDS envisioned by its developer. During run-time it compares a database of regular expressions that define suspicious log file strings against Syslog file entries and reports any matches as alerts [361]. Classification of these alerts include informational, warning, and alert detected. Output is both written to standard out and a log file is generated. Analysis by Guard26 is limited to Syslog file types, and only those suspicious log file strings found within its database. This concept to provide a real-time IDS utilizing this method has excellent potential, if implemented with real-time correlation. Another idea could consist of real-time correlation of the alerts generated by this tool with the web server log events.

3.3.3 Osiris and Samhain

HIMSs periodically scan one or more host systems for modifications to files or their attributes and record variations or unmodified entries to result logs. They do this to maintain an audit trail of all of the changes to a system from a particular baseline. Osiris [196] and Samhain [385] as HIMS are structured very differently. They cannot be directly compared as they are used in different scenarios, yet they maintain the shared title of the two most widely used HIMS.

Osiris maintains an event log of changes to the file system including files and their attributes, user and group lists, resident kernel extensions or modules, network ports, other non-module specific kernel elements, and administrator services [389, 390, 196]. For maximum operating efficiency that ensures both ease of use and the maintenance of the integrity of events, Osiris is deployed in a securely distributed fashion employing security devices such as digital certificates and the use of the SSL to protect the integrity and privacy of all communications between its components [390]. During its operation, Osiris does not base its logic upon the use of signatures, but takes periodic scans of the system storing the results in a centrally located and managed Berkeley [290] database. All of the scan data and log messages are neither signed nor encrypted during storage, which requires the administrator to ensure a hardened and secure system for the management console [389]. Inherently advantageous to Osiris are its abilities to deploy and manage from either a Windows or Unix-like environment, and to provide additional functionality through the development of modules. With respect to noise, regular expressions implement filtering capabilities on Osiris. Filtering consists of preventing matched events from being logged. A drawback to this tool presents itself when a regular expression is applied that prevents significant events from being logged [389].

Similarly, Samhain albeit limited to use only on Linux systems, expands in two ways to other areas of integrity monitoring. The first way is the ability to monitor the kernel itself on Freebsd [320] and Linux systems, and the second way is that the management consoles can be accessed and modifications made through a locally hosted web application console [389]. In relation to kernel monitoring, Samhain has the capability of monitoring the interrupt handler, the system call table, parts of the virtual file system layer and a portion of each system call handler [389]. These capabilities enable Samhain to effectively detect rootkit and other malicious kernel modifications. The tool Samhain was originally developed as a host-based IDS that would monitor many disparate aspects of an environment and utilizes integrity monitoring as a means of and in addition to other means of detecting an intrusion. It includes methods to provide file integrity checking, log file monitoring, rootkit and rogue Set User IDentification (SUID) [34] executables detection, mount point modifications, port monitoring and hidden processes [385]. During operation Samhain can be run as a standalone system, or it can be centrally managed on a variety of operating systems including those that are Linux or Unix based, and can be run under Cygwin [217] on Windows. The Windows and Cygwin configuration for running Samhain is not optimal. The reason for this is that security cannot be ensured between threading, since during development this tool was not designed for use on a Windows system. It therefore, may overlook areas where intrusions may be detected more quickly such as the registry [92], or the Windows kernel space.

Methods to prevent tampering and detection of running Samhain or Osiris may include one or more of the following actions: signing both databases and configuration files with Pretty Good Privacy (PGP) [258] signatures, stealthy operation of the program by hiding its execution from the process list, and obfuscation of the executable and configuration file locations within alternate data streams, steganographically within non-related files or within file padding [385, 389]. With the output logged from these HIMSs, this provides an investigator an entire audit trail of the system starting from its specified baseline. Both systems include an extensive log event collection definition, which aid in both parsing and analysis of the events logged by either system. The log files that they generate are not only a point of correlation and cross-correlation, but they verify the integrity of other logs. Other than IDSs, these logs were included in development design and are not only used by developers for debugging. This means that they include definitive meanings for certain events that these tools inherently comprehend. Alone they do not provide the analysis of log files, however the addition of events coming from a HIMS would provide the necessary corroboration as well as a more thorough detail of events for the investigation of an incident within a forensics perspective.

3.3.4 OSSec

OSSec [298] is an open source, host-based IDS that aside from log analysis performs active response, monitoring of file integrity and policy changes, real-time alerting, and rootkit detection. This multiplatform tool runs on Linux, OS X, Solaris, HP-UX [201], Advanced Interactive eXecutive (AIX) [206] and Windows. Log inspection is performed by the `logcollector` and `analysisd` processes. They perform collection and parsing, decoding, filtering, and classification analysis on the specified log files. The log file formats that are supported include process or history log, Syslog, NCSA, and W3C ELF for HTTP, FTP and Simple Mail Transfer Protocol (SMTP) [238], MS event log format (Application, Security, System) and IIS web server default log format [298]. This tool assists with Payment Card Industry (PCI) [305] compliance, in that it enables the real-time monitoring of log files, but cannot perform correlation between different sources. Due to the nature in which the process history information is stored, a plugin that performs correlation between process history and log event entries could be developed in the future.

3.3.5 Snort

As a functionality requirement, an IPS must provide the interception of all network communication, which ensures the capability to halt or block, and manipulate or sanitize incoming malicious traffic. The multiple modes attributed to Snort [263] include one that provides the functionality of a network IPS, one that functions as a network IDS, and the last is a sniffer device that provides streamed or logged data [39]. Snort provides the capability of investigating the network at the packet level. It uses rules or signatures to define troublesome events, and perform actions on these events in addition to the generation of packet log files. The program itself does not include a GUI, and can be linked to one to provide a more overall viewpoint. The output of Snort includes a very basic reporting style that can be manipulated externally, but this manipulation requires a separate program to reveal a legible report [263]. Snort has the capability to provide lower network level events to a log file, MySQL database, or standard output which could provide an excellent event cross correlation source. This program is also capable of reading previously generated network packet dumps and performing analysis on them. It requires a comprehensive rule set adapted to individual networking configurations to function with the greatest efficiency and the least number of false positives as with any IDS or IPS. Pertaining to forensics however, Snort does not provide the inherent functionality of correlation, appropriate reporting and visual interface required of a forensics tool as it was not designed to fulfill this purpose. In conclusion, this open source project does provide an excellent example of signature, protocol and anomaly-based packet event detection methods, which should be adapted into an effective forensic system.

3.3.6 Splunk

Search engines provide the necessary means of navigation through the vast series of electronic tubes that comprise the Internet. Similar in functionality to a search engine, Splunk [354] indexes data from any type of log source and system specific dataset. Splunk employs configuration files, registry entries and loaded modules or drivers to provide a structured and integrated search method. Splunk provides real-time access, analysis, integrity, change monitoring, reporting and can operate autonomously [354]. Splunk is implemented by an outward facing web interface for interaction with its internal engine. It does not utilize an external database instance, and instead relies on buckets and indexing for flat file database construction with automatic log type detection and event parsing [353]. As a forensics tool, Splunk provides the ability to verify the integrity of the data, provide statistical analysis and reporting, and the ability to provide evidentiary correlation between data sources, configurations, and system information all in real-time. Splunk handles all log data in an efficent manner, and yet still requires a lot of configuration on the part of the user.

3.4 Conclusion

The three categories of industry leading tools examined within this comparison represent application specific, development framework, and intrusion detection or host integrity monitoring log solutions. The conclusions regarding state of the art forensic log solutions outline potential problems as well as requirements that when instantiated would constitute an effective analysis solution. One issue made apparent was that many tools provide summaries of information that might remove or cause specific events of evidentiary value to a forensic investigator to be passed over. The main summary of the comparison within this chapter is showcased in Table 3.4. The functional and non-functional requirements outlined previously, consist of log management, integrity verification, timeline generation, legible reporting, extensibility, scalability, and appropriate documentation. Appropriate methods of log management include log file rotation, a central stealth repository for the log files and obfuscation of the analysis tools themselves. The importance of integrity in a forensic log analysis tool is of utmost priority when dealing with evidentiary material. Integrity is maintained through proper chain of custody procedures, compliance with applicable standards to the incident scenario, and the instantiation of a variety of correlation techniques to provide corroboration. The three types of event correlation include normal correlation between multiple events within a single log file, cross-correlation of events from separate log sources, and cross-correlation of events to external source data structures. The forensic reconstruction of an incident requires the generation of a timeline of significant events prior to and occurring within the specified duration. Significant events include records of command execution (found within process accounting), log on and log off statistics, and events that signify attacks. It is a forensic investigators duty to provide meaning to the raw data presented as evidence. To ensure understanding of the investigators interpretation, a forensic log analysis tool should incorporate legible reporting. This legible reporting should incorporate graphical representations of findings and generate at least two styles of documents. The first document style should be clear, concise, comprehensive and incorporate terms applicable in a legal setting for court officials, and the other document should provide a generalized format of the same data contained within the first for a wider audience. Due to the tendency of log files to contain large amounts of data, the scalability of an analysis solution is important. Methods to deal with large log file size include the use of database technologies and the choice of programming language which should provide both operating system independence and efficiency. Extensibility highlights the importance of modularity, as modules provide the capability of specifying different events according to the incident under investigation. To maximize acceptance and potential of a forensic analysis tool with respect to the end user, as with any software engineering project, comprehensive documentation is essential. An initial list of over 100 tools covering the vast spectrum of current log analysis implementations was discovered and investigated during the course of our research. Due to space requirements, other tools we were made aware of during our investigation were not included in the comparison. They are noteworthy here since they pertain directly to forensics and web application security. Forensic toolkits such as FTK [3], Encase [197], and Sleuthkit [42] are the three main forensic toolkits used by incident responders. All three lack in log analysis functionality beyond standard viewing and search capabilities. Web application vulnerability scanners such as Paros [61], Nikto [362], and Webscarab [150] can be used to generate attack signatures within the log files, but do not perform log analysis themselves.

Table 12: Log Analysis Tools Comparison

	Tools	Descriptions										
		Standards Compliance	Performs Correlation	Correlation Data Source	Admissibility†	Multiple Platform	Useability	Reporting	Scalability	Real-Time	Compression	Generation of Alerts
Application Specific	Psacct	N	N	Y	-	Y [i]	CL	-	Y	Y	N	Y [b]
	Acctsum	N	N	N	-	Y [i]	CL	Y [H]	Y	N [c]	N [t]	N
	Analog	N	N	N	-	Y [ii]	CL	Y [H,e]	Y	N [c]	N	N
	Anteater	N	N	N	-	Y [iii]	CL	Y [H,e]	N	N	N	N
	Awstats	N	N	N	-	Y [iv]	B	Y [S,P,H]	N	Y [f]	N	N
	Breadboard Bi Web Analytics	N	N	N [v,h]	-	Y [iv]	B	Y [P,H,X,R]	Y	Y	N	N
	Calamaris	N	N	Y	-	Y [i]	CL	Y [H]	Y	N [c]	N	N
	Chklogs	N	N [a]	N	-	Y [i]	CL	N	N	N [c]	Y [rt]	N
	CORE Wisdom	N	N [j]	N	-	N [v]	EUI	Y [g]	Y	Y	N	N
	Eventlog Analyzer	Y	Y [SF]	N	-	Y [iv]	B	Y [H,P,C]	Y	Y	N	N
	Ftpweblog & Wwwstat	N	N	N	-	Y [i]	CL	Y [H]	Y	N [c]	N	N
	Funnel Web® Analyzer	Y	N	N	-	Y [ii]	CL	Y [P,H,E,R,W]	Y [$]	Y [$]	N	N
	HTTP-Analyze	N	Y [SF]	N	-	Y [iii]	CL	Y [H]	N	N [c]	Y [rt]	N

Continued ...

	Tools	Descriptions										
		Standards Compliance	Performs Correlation	Correlation Data Source	Admissibility†	Multiple Platform	Useability	Reporting	Scalability	Real-Time	Compression	Generation of Alerts
Application Specific	logjam	Y	N	Y	-	N [v]	B	Y [H]	N	N	N	N
	Logparser	Y	N [a]	N	-	N [v]	N[EXT]	Y [*]	Y	N	N	N
	Lire	Y	N	N	-	Y 1	CL	Y[H,X]	Y	N	N	Y
	Logrep	Y	N	N	-	Y [ii]	CL	Y [H]	Y	Y	N [xs]	N
	Logstalgia	N	N	N	-	Y [ii]	EUI[g]	N	Y	Y [c]	N	N
	Logsurfer	N	Y	Y	-	Y [i]	CL	Y	Y	Y	N	Y
	Swatch	N	Y	Y	-	Y [i]	CL	N	Y	Y	N	Y
	Tenshii	N	N [a]	Y	-	Y [i]	CL	Y [C]	Y	Y	N	Y
	Mywebalizer	N	N	Y	-	Y [ii]	CL	Y [H]	Y	N	Y	N
	Open Web Analytics	N	N	N	-	Y [ii]	B	Y [H]	Y	Y	N	N
	Pyflag	Y	N [a]	Y	-	Y [ii]	B	Y [H]	Y	N	N	N
	Sawmill	Y	N [a]	Y	-	Y [ii]	B	Y [H]	Y	Y	Y [xs]	N
	Squidj	N	N	Y	-	Y [i]	CL	Y [S]	Y	N [c]	N	Y
	Scansquidlog	N	N	Y	-	Y [i]	CL	Y [S]	Y	N [c]	N	N
	Visitors	N	N	Y	-	Y [ii]	CL	Y [H]	Y	N	N	N
	Weblogmon	N	N	Y	-	Y [iii]	CL	N [F]	Y	Y	N [xs]	Y

Continued . . .

	Tools	Descriptions										
		Standards Compliance	Performs Correlation	Correlation Data Source	Admissibility†	Multiple Platform	Useability	Reporting	Scalability	Real-Time	Compression	Generation of Alerts
Development Tools	ApacheDB	Y	N [a]	N	-	Y [ii]	D	D	Y	D	N	D
	Mod_log_SQL	Y	N	Y	-	Y [ii]	D	D	Y	D	N	D
	Cascade Software	Y	N [a]	N	-	Y [ii]	D	Q	Y	Y	N	Y
	Crystal Reports	Y	N [a]	N	-	N [v]	EUI	Q	Y	Y	D	D
	JasperReports	N	N [a]	N	-	Y [iii]	EUI	Q	Y	Y	D	D
	SEC	N	Y [a]	N	-	Y [iii]	CL	D	Y	N [c]	N	D
IDS and HIMS	ArcSight Logger and ESM	Y	Y [a]	Y	-	N [A]	B	Y [r,g]	Y	Y	Y	Y
	Guard26	N	N	Y	-	Y [i]	CL	N	N	Y	N	Y [b]
	Osiris	Y	N [j]	Y	-	Y [ii]	CL/B	N	Y	Y	N	Y [b]
	Samhain	Y	N [j]	Y	-	N [vii]	CL	N	Y	Y	N	Y [b]
	OSSEC	N	N [a,j]	Y	-	Y [ii]	CL	N	Y	Y	N [xs]	Y [b]
	Snort	N	N	Y	-	Y [ii]	CL	Y [uf]	Y	Y	N	Y
	Splunk	N	N [a,j]	Y	-	Y [ii]	CL/B	Y	Y	Y	D	Y

[i] OS X, Unix, Linux, FreeBSD

[ii] Any operating system, Windows & OS X (pre-built binaries), Unix, Linux, FreeBSD

[iii] Interpreted language OS independent

[iv] Web application, requires a server, any OS browser is used

[v] Windows only

[a] Does grouping or enables correlation development

[b] Not Syslog, own process log

[c] Can be scripted

[e] Statistics

[f] Dynamic and static

[g] Enhanced graphics

[h] Data mining

[j] Provides mechanisms for verifying integrity

[t] Truncation

[r] Utilizes a proprietary language mixing XML and SQL to script report generation

[$] Commercial version

[D] Analyst must code for functionality of individual projects

[Q] Developer generated queries to create reports

[F] Raw output is organized by users in flat files only

[H] HTML

[P] PDF

[W] Word

[C] CSV

[E] Excel

[X] XML

[R] RTF

[S] Standard Out

[uf] Unformatted raw output

[rt] Rotation

[xs] Extracting selections of logs

[EUI] Executable user interface

[SF] Correlation within the same file

[EXT] External Program provides GUI

[CL] Command Line

[B] Browser

[N] No

[Y] Yes

[*] CSV, TSV, XML, Syslog, W3C, IIS,SQL, DATAGRID CHART and NAT

[†] Attempting to determine the admissibility of current log analysis tools via forensic precedence within case law is a difficult procedure. A search of Canadian and American case law within the Lexis-Nexis [253] and Canlii [136] legal databases yielded only references to forensic toolkits such as Encase [197] or Forensic ToolKit (FTK) [3] which retain rudimentary log analysis functionality. Also, in court proceedings where logs were used as definitive evidence, records did not specify the log analysis tool names that investigators employed.

Chapter 4

Attack Detection Using Regular Expressions

This chapter is devoted to descriptions of the attacks that our forensic log analysis program can detect. For each type of attack we will provide the regular expression used for its detection along with a description of the category it belongs to. What follows is an introduction to the exploitation of web applications describing the most commonly desired outcomes for an attacker. As such, our methodology is limited to those web applications that use the AWS as a basis, since the target log file that can be analysed is the AWS `access_log` in CLF or ELF. Regular expression extensions for other log file format contents can render similar patterns for the discovery of attacks. Next, an overview of our methodolgy utilizing `Regex`s is explored. The `Regex` knowledge base that we employed is then approached, examined, and demonstrated in detail providing a description by attack or outcome as specified. Outlining the group description is an introduction, attack scenario, and attack examples with the regular expressions that detect them from our tool. One attack string example is given to demonstrate the functionality of the regular expression. This sample string may be of use multiple times for separate regular expressions if the string would trigger during an attack scenario. Most injection attacks are application specific, such that on the client-side it may affect a specific browser and version or web application that runs client-side, whereas on the server-side, it may be affecting service versions or web applications that run as services.

4.1 Exploiting Web Applications

Web application vulnerabilities can exist in both server-side and client-side applications. The methods of attack vary according to the desired results of the attacker. Once an application is vulnerable to a specific method of attack, it provides an attacker a variety of exploitational outcomes. Desired exploits could include code execution, DoS, information disclosure, defacement, session hijacking, loss of trust, or data manipulation. Code execution could happen on the server-side, exposing not only server processes, but also any data resident on the server. When code execution happens on the client-side, user-level data can be accessed such as browser history or other files contents and code that attempts to gain elevated priviledges may then be executed. Information disclosure severity can range from the access of trivial data to the disclosure of confidential or proprietary property. Examples of information disclosure include traversal or the mapping of the directory tree structure on the server, viewing deprecated development versions of applications, or revealing confidential data such as log files, proprietary company or government secrets. The information disclosed can be stolen, copied, or possibly modified of that data. Causes of information disclosure include misplaced configuration files, tapping of secure sessions, and a wide variety of scripting attacks.

4.2 Methodology

In order to analyze AWS `access_log` files and detect potential attack events, we rely on regular expressions to describe web application attack scenarios. The type of web application vulnerabilities detected are related to input validation. The applicable regular expressions were gathered and compiled from two main sources, which are the "OWASP Top Ten attacks from 2007" [267] and the PHP:Hypertext Preprocessor Intrusion Detection System (PHPIDS) default filter file [192]. In total, 80 regular expressions are compiled to define matches in AWS `access_log` files for particular web application attack scenarios. Describing the attacks as regular expressions enables the extendability and maintainability of our approach if new attacks are discovered or the described attack scenarios are revised. In order to put this approach into practice, we implemented a tool that forensically detects attacks against web applications. Our tool's features and functionalities are described in detail within Chapter 5. In the next section, we provide a set of examples of detected attack scenarios with their corresponding regular expression.

4.3 Injection Attacks

One variety of attack that exploits this vulnerability are known as injection attacks. Injection attacks leverage the modification or provision of data supplied to an application which may be designed maliciously to execute specific system commands or code fragments [300]. In this section, we will introduce HTTP header injections and provide an explanation of SQL injections and give the regular expression examples that our tool is capable of detecting for this type of attack. Current browsers do not limit the variety of code they accept, execute or render for end users. To round out this section, we present the examples of the other code and protocol related injections that can be detected using our tool.

4.3.1 Protocol-Related Injection

Two applicable protocol related injection attacks that we consider to be influential to the development of our tool are HTTP header injection and Light-weight Directory Access Protocol (LDAP) [233] URL injection. The following paragraphs explain a future additional correlation source and give an example of how to detect using our forensic tool an injection attack attempt.

Manipulation of HTTP headers at the simplest level can enable an attacker to masquerade as originating from a different source, or to request a specific version of the intended target should a web application dispense data based on browser validation [338]. More complicated attacks can also be performed, such as those first outlined in [236] and [237], whereby an attacker could split HTTP responses, perform session fixation based upon access to or modification of the user cookie portion of the header, perform a Cross Site Request Forgery (CSRF) attack by modifying the address of the header or a XSS attack upon an unsuspecting victim. Unfortunately the AWS only records a portion of the header information when logging events to the `access_log`. Thinking forensically about detecting this type of attack, one future application could be made by logging the HTTP headers on the server and correlating those contents to AWS log events for multiple simultaneous requests.

Internal facing instantiations of the LDAP on an attackers target network may be accessible through the browser by injection if request input has not been appropriately validated. LDAP enables users to access a variety of information located within storage directories. In most deployments, each resource has a dedicated directory and corresponding data pertaining to its individual directory within the network tree structure [105].

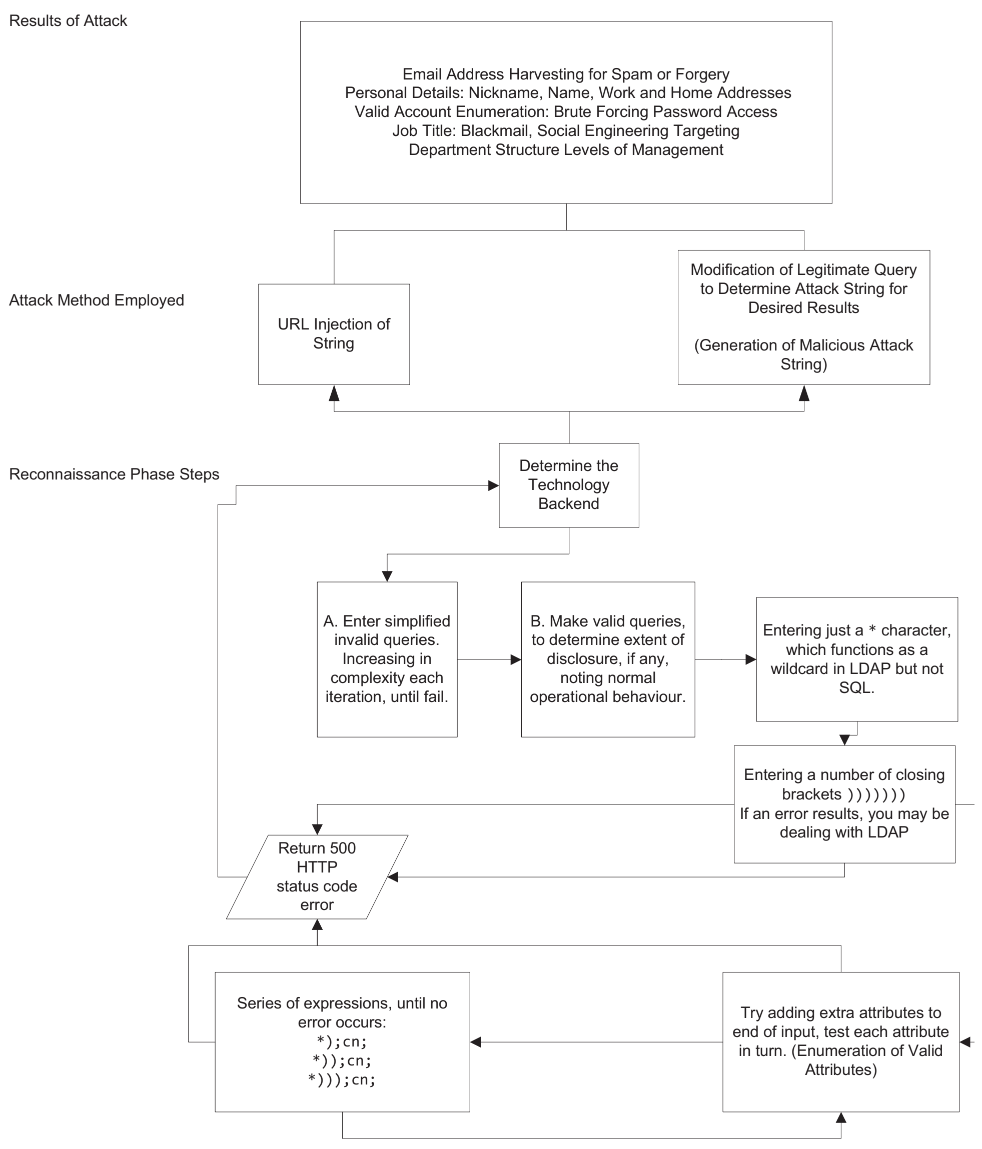

Results of Attack
Email Address Harvesting for Spam or Forgery
Personal Details: Nickname, Name, Work and Home Addresses
Valid Account Enumeration: Brute Forcing Password Access
Job Title: Blackmail, Social Engineering Targeting
Department Structure Levels of Management
Attack Method Employed
URL Injection of String
Modification of Legitimate Query to Determine Attack String for Desired Results
(Generation of Malicious Attack String)
Reconnaissance Phase Steps
Determine the Technology Backend
A. Enter simplified invalid queries. Increasing in complexity each iteration, until fail.
B. Make valid queries, to determine extent of disclosure, if any, noting normal operational behaviour.
Entering just a * character, which functions as a wildcard in LDAP but not SQL.
Entering a number of closing brackets))))))) If an error results, you may be dealing with LDAP
Return 500 HTTP status code error
Series of expressions, until no error occurs: *);cn; *));cn; *)));cn;
Try adding extra attributes to end of input, test each attribute in turn. (Enumeration of Valid Attributes)

Figure 5: LDAP Injection Scenarios [360]

Regular Expression to Detect LDAP Injections

Figure 5 demonstrates scenarios in which an attacker employs LDAP injections to procure given results, and methods of probing an application in preparation to the exploit. The following example detects requests made that attempt to exploit potential LDAP vectors within an event.

Regular Expression String ($Regex_{64b}$):

```
(?:\|\(\w+=\*)|(?:\*\s*\)+\s*;)
```

Detected Attack String:

```
ldap://ldap.example.net:2600/o=Concordia\%20University,c=CAN??sub?(cn=Rick\%20Astley)
```

Description:

This would detect a URL of the form `ldap://host:port/DN?attributes?scope?filter?extensions` specified from "RFC 4516" [260] to utilize LDAP from a browsing perspective. This vector may be exploited for the purposes of information disclosure in the form of detailing the network structure, accessing files, describing directory structure which may leak personnel data, or performing directory traversal. A more detailed description of this type of attack is presented in Section 4.3.3.

4.3.2 SQL Injection

Proper utilization of a data store is beneficial for both efficiency and organziation in the development of any application and web applications are no exception. An SQL injection attack targets vulnerable data access code [266]. A web application that transposes user supplied data into an SQL query is susceptible if this data has not been sanitized due to every SQL implementations lack of distinction between the control and data planes [57, 48]. Thus, an attacker would send SQL input, executed for example by inserting a meta-character that alters an intended query or inserts a new query into the application [266, 57]. Effective attacks instantiated by attackers are performed in order to modify the database, disclose or destroy critical information, tamper existing data, execute external operating system commands, along with a variety of other outcomes. The following examples are a sample of the regular expressions we deploy in our tool to detect SQL injection attempts.

The demonstration of scenarios in which an attacker utilizes SQL injections are conveyed in Figure 6. It also outlines preliminary detection of SQL vulnerability interactions performed before an actual attack request.

Detect Conditional SQL Injections

Regular Expression String ($Regex_{41}$):

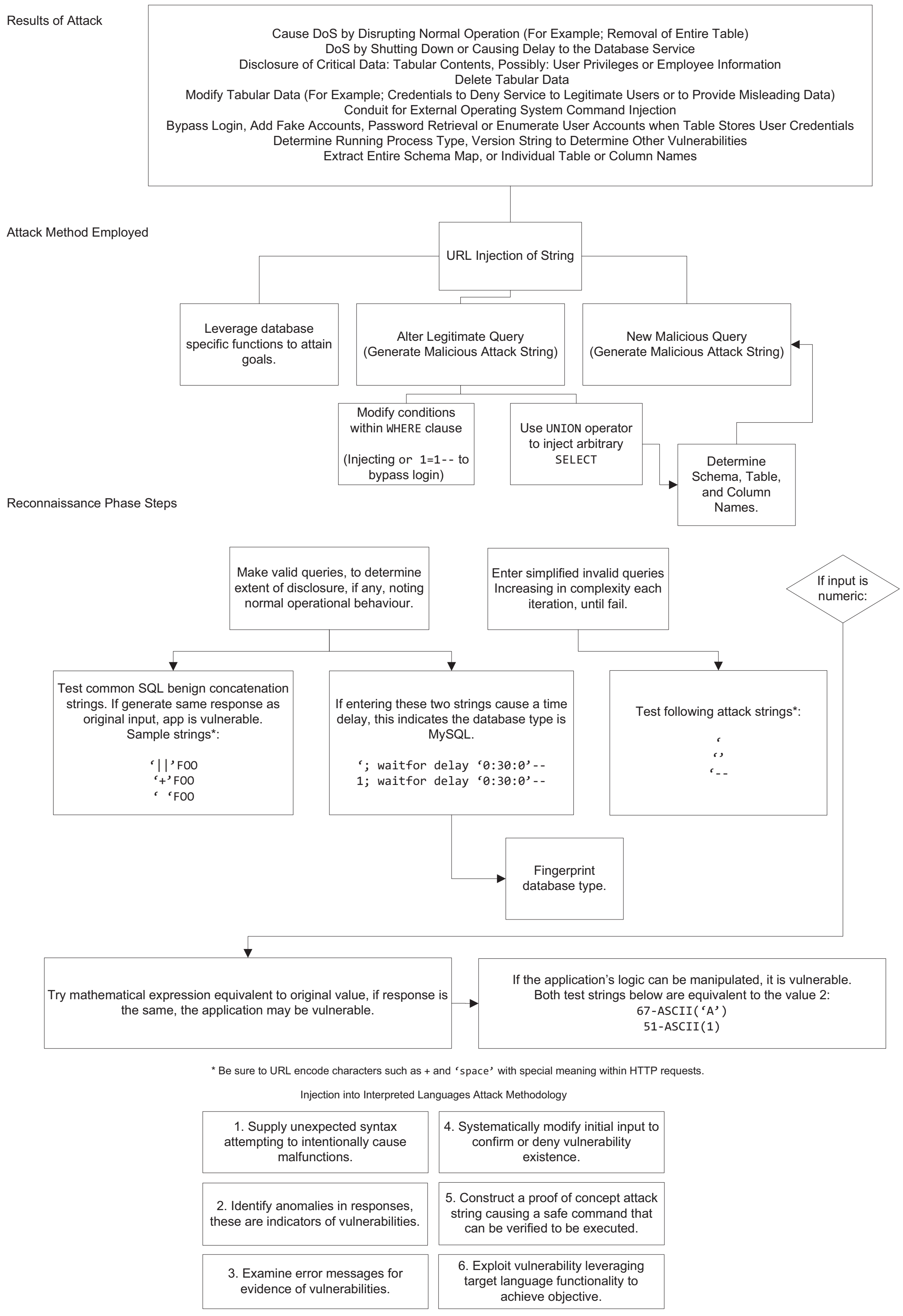

Results of Attack
Cause DoS by Disrupting Normal Operation (For Example; Removal of Entire Table)
DoS by Shutting Down or Causing Delay to the Database Service
Disclosure of Critical Data: Tabular Contents, Possibly: User Privileges or Employee Information
Delete Tabular Data
Modify Tabular Data (For Example; Credentials to Deny Service to Legitimate Users or to Provide Misleading Data)
Conduit for External Operating System Command Injection
Bypass Login, Add Fake Accounts, Password Retrieval or Enumerate User Accounts when Table Stores User Credentials
Determine Running Process Type, Version String to Determine Other Vulnerabilities
Extract Entire Schema Map, or Individual Table or Column Names
Attack Method Employed
URL Injection of String
Leverage database specific functions to attain goals.
Alter Legitimate Query (Generate Malicious Attack String)
New Malicious Query (Generate Malicious Attack String)
Modify conditions within WHERE clause (Injecting or 1=1-- to bypass login)
Use UNION operator to inject arbitrary SELECT
Determine Schema, Table, and Column Names.
Reconnaissance Phase Steps
Make valid queries, to determine extent of disclosure, if any, noting normal operational behaviour.
Enter simplified invalid queries Increasing in complexity each iteration, until fail.
If input is numeric:
Test common SQL benign concatenation strings. If generate same response as original input, app is vulnerable. Sample strings*: '||'FOO '+'FOO ' 'FOO
If entering these two strings cause a time delay, this indicates the database type is MySQL. '; waitfor delay '0:30:0'-- 1; waitfor delay '0:30:0'--
Test following attack strings*: ' '' '--
Fingerprint database type.
Try mathematical expression equivalent to original value, if response is the same, the application may be vulnerable.
If the application's logic can be manipulated, it is vulnerable. Both test strings below are equivalent to the value 2: 67-ASCII('A') 51-ASCII(1)
* Be sure to URL encode characters such as + and 'space' with special meaning within HTTP requests.
Injection into Interpreted Languages Attack Methodology
1. Supply unexpected syntax attempting to intentionally cause malfunctions.
2. Identify anomalies in responses, these are indicators of vulnerabilities.
3. Examine error messages for evidence of vulnerabilities.
4. Systematically modify initial input to confirm or deny vulnerability existence.
5. Construct a proof of concept attack string causing a safe command that can be verified to be executed.
6. Exploit vulnerability leveraging target language functionality to achieve objective.

Figure 6: SQL Injection Scenarios [360]

```
(?:\)\s*like\s*\()|(?:having\s+[\d\w\-"]+\s*[(=<>~])|
(?:if\s?\([\d\w]\s*[=<>~])
```

Possible Detected Attack String:

```
SELECT * FROM users WHERE username = '' having 1=1--
```

Description:

A conditional SQL injection relies on the insertion of an SQL statement that includes specific keywords such as `like` and `having`. In $Regex_{41}$, the group `(?:having\s+[\d\w\-"]+\s*[(=<>~])` attempts to match the keyword `having` in combination with a variable amount of whitespace, possibly a number which is followed by an equal sign =. `=` follows. This results in the sub-string `having 1=` to match. The above attack statement is equivalent a scenario where a username `' having 1=1--` is fed to the statement:

```
SELECT * FROM users WHERE username = 'x' and password = 'y'
```

This attack causes the database to return an error message similar to the following [360]:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e14'
[Microsoft][ODBC SQL Server Driver][SQL Server]Column 'users.ID' is
invalid in the select list because it is not contained in an aggregate
function and there is no GROUP BY clause.
```

As shown the error discloses the relevant table name `users` and its first column name `ID`. Leveraging an attack such as this as further input into future attack attempts as indicated in [360], an attacker can determine the entire tabular structure. The column names from the `users` table may correspond to valid usernames this attack. If so, an attack of this type will disclose information vital to a subsequent attack capable of modifying data within the table.

4.3.3 Operating System Command Injection

Through the use of a target systems dependancies such as specific languages or services, an attacker may instigate the execution of operating system level commands. Upon successful injection, specified commands will run with the same permissions as the user object running the injection entry point

[14]. Using previous knowledge of a target operating system enables an attacker to specify access to a path directly through injection. This first sample regular expression enables our tool to detect the path traversal to complete an attack that specifically targets an operating system executable or directory.

The following methods of attack that are described demonstrate the capability of our forensic log tool to detect two forms of operating system command injections. Both of these examples perform directory traversals. Operating system command injection scenarios, and probing methods of their detection are described within Figure 7.

Detects Specific Directory and Path Traversal

Regular Expression String ($Regex_{11}$):

```
(?:%c0%ae\/)|(?:(?:\/|\\)(home|conf|usr|etc|proc|opt|s?bin|local|dev|
tmp|kern|[br]oot|sys|system|windows|winnt|program|%[a-z_-]{3,}%)
(?:\/|\\))|(?:(?:\/|\\)inetpub|localstart\.asp|boot\.ini)
```

Possible Detected Attack String:

```
http://www.example.com/doc/..%5c../Windows/System32/cmd.exe?/c+dir+c:\
```

Description:

The Regular Expression $Regex_{11}$ detects specific path traversal attacks which target a number of both critical and non-critical system directories. As an example, this URL request is used to traverse directories for the purposes of command execution. Upon successful submission to the server, an attacker procures the full listing of files located in the root directory `C:\`. It launches the `cmd.exe` command shell and executes the command `dir c:\` in the shell. The `%5c` represents the ASCII escape code for the backslash character \. Due to the many permutations of this attack, another similar example might include a well-known directory typically vulnerable to the traversal attack; the scripts directory of IIS.

Directory Traversal

Directory traversal is performed through the injection of an operating system command. Since it is structured similarly across all operating systems, and the risks associated with it vary widely other

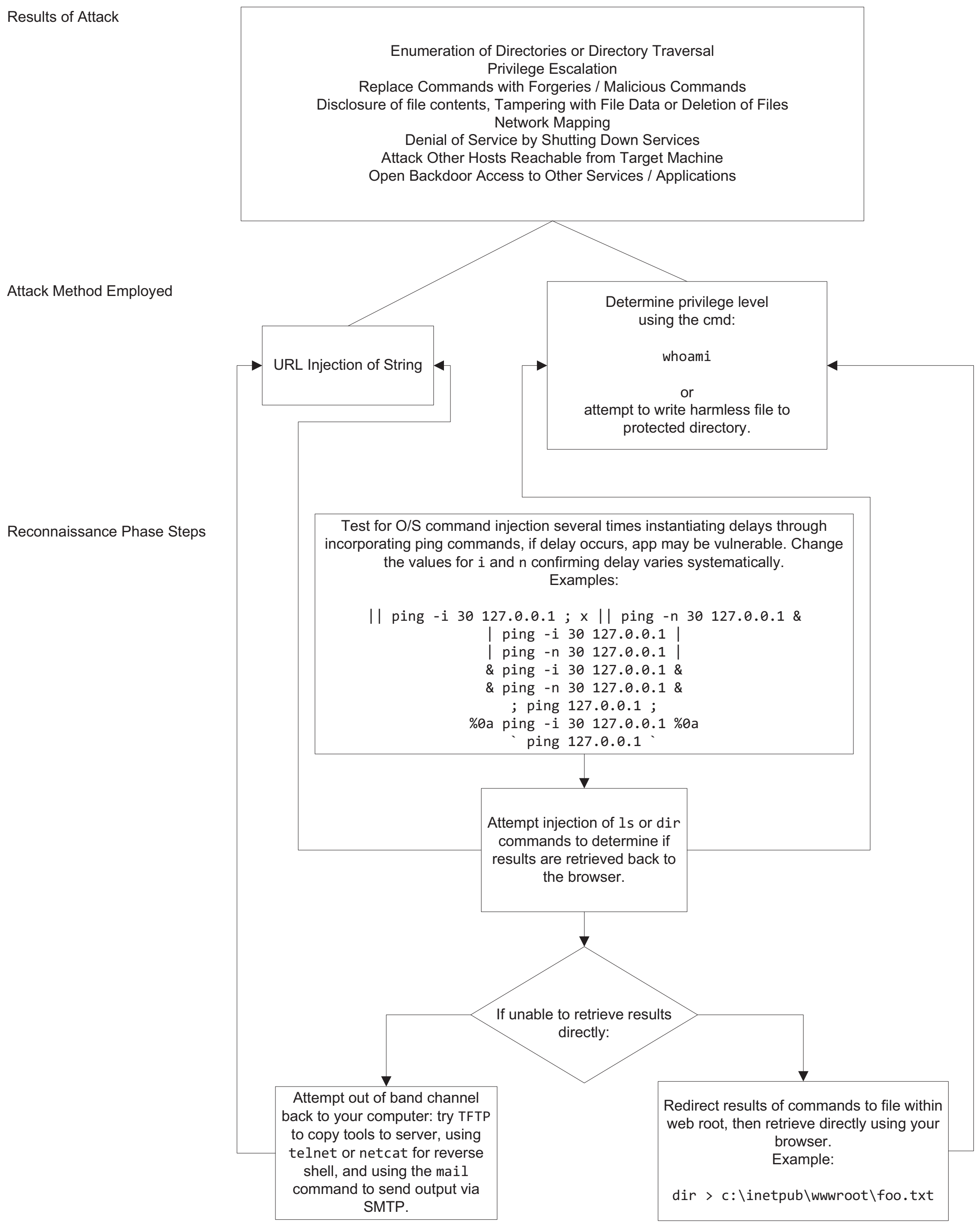

Results of Attack
Enumeration of Directories or Directory Traversal
Privilege Escalation
Replace Commands with Forgeries / Malicious Commands
Disclosure of file contents, Tampering with File Data or Deletion of Files
Network Mapping
Denial of Service by Shutting Down Services
Attack Other Hosts Reachable from Target Machine
Open Backdoor Access to Other Services / Applications
Attack Method Employed
URL Injection of String
Determine privilege level using the cmd: whoami or attempt to write harmless file to protected directory.
Reconnaissance Phase Steps
Test for O/S command injection several times instantiating delays through incorporating ping commands, if delay occurs, app may be vulnerable. Change the values for i and n confirming delay varies systematically. Examples:
|| ping -i 30 127.0.0.1 ; x || ping -n 30 127.0.0.1 &
| ping -i 30 127.0.0.1 |
| ping -n 30 127.0.0.1 |
& ping -i 30 127.0.0.1 &
& ping -n 30 127.0.0.1 &
; ping 127.0.0.1 ;
%0a ping -i 30 127.0.0.1 %0a
` ping 127.0.0.1 `
Attempt injection of ls or dir commands to determine if results are retrieved back to the browser.
If unable to retrieve results directly:
Attempt out of band channel back to your computer: try TFTP to copy tools to server, using telnet or netcat for reverse shell, and using the mail command to send output via SMTP.
Redirect results of commands to file within web root, then retrieve directly using your browser. Example:
dir > c:\inetpub\wwwroot\foo.txt

Figure 7: Operating System Command Injection Scenarios [360]

than access to system commands, it was placed under this individual category. Simple traversal enables an attacker to map out the structure of the intended target with respect to applications. It may expose those applications that are installed or running, as well as structural information, configurations, and data files. The preceeding example detected system-specific directories, whereas the following regular expression example can detect more generalized directory traversal attacks.

Detects Basic Directory Traversal

Regular Expression String ($Regex_{10}$):

```
(?:(?:\/|\\)?\.+(\/|\\)(?:\.+)?)|(?:\w+\.exe\??\s)|(?:;\s*\w+\s*\/
[\w*-]+\/)|(?:\d\.\dx\|)|(?:%(?:c0\.|af\.|5c\.))|(?:\/(?:%2e){2})
```

Possible Detected Attack String:

```
../../../../../Windows/system.ini
```

Description:

$Regex_{10}$ is used to detect directory traversal attacks. Examples of other matched sub-strings are, `/../..` and `/../`. Within this regular expression, the above attack string is captured by the grouping `(?:(?:\/|\\)?\.+(\/|\\)(?:\.+)?)`. The Table 13 below describes the process of the regular expression match for this grouping:

Table 13: Regular Expression Matching for $Regex_{10}$

Step	Regex	Input	Description
1	`(?:\/|\\)?`		Optional pattern, no match is found
2	`\.+`	`..`	Matching the two dots
3	`(\/|\\)`	`/`	Matching the "/" symbol
4	`(?:\.+)?`	`..`	Matching two dots

The directory traversal exploit utilizes the HTTP protocol for delivery and it enables the access of restricted directories and the execution of external commands outside of a target web server's root directory. The only tools required by an attacker in order to perform this exploit is a method of sending HTTP requests, such as a web browser, and the relative knowledge to determine the location of pertinant files and directories on the system applicable to the desired attack outcome. The following URL demonstrates one version of this exploit:

```
http://www.example.com/example.asp?display=userpage.html
```

Using this URL to access the web application, the browser requests the dynamic page named `example.asp` from the server. This request also sends the parameter `display`, which has the value set to `userpage.html`. Upon recieving this request, the server's dynamic page, the `example.asp` page retrieves the indicated file `userpage.html` from the server's file system. The retrieved page is then rendered and sent back to the requester's browser for display. For this case, a malicious user assumes that `example.asp` can retrieve any file from the target file system and attempts to send a crafted URL such as this one:

```
http://www.example.com/example.asp?display=../../../../../Windows/system.ini
```

During the execution of a successful exploit, requesting the above URL causes the dynamic page to retrieve the file `system.ini` from the file system and render it back for the user. However, to a vulnerable application running as a priveledged user, an attacker is not limited to specific directories and could succeed in traversing multiple directory tree levels in order to find any desired file. For example a target might include the `/Windows`, `/System` or `/SYSTEM32` folders on the server.

4.3.4 Local or Remote File Inclusion

An exploit that uses or prevents the use of a library file or separate executable file from the execution of code can be performed against certain web applications. One example of this could include a scenario in which an intruder forces the web server to execute malicious code contained within a file by first injecting directive information to the application, which executes code by command injection to include the previously uploaded or remotely hosted code [10]. Being very prevalent on the Internet, the PHP language is inherently vulnerable to this type of attack as it is capable of pulling in a local file and running it from a remote command [335]. As a proof of concept, we provide in the following two examples of attack detection. The first one shows the inclusion of the `passwd` file on a Unix based system through the PHP attack vector. The second example shows an attempted exploit through the `firefoxurl` attack vector. The Figure 7 describes local and remote file inclusion vulnerabilities. It also depicts typical actions made by an attacker to determine effective exploitational avenues.

Detects `passwd` File Inclusion

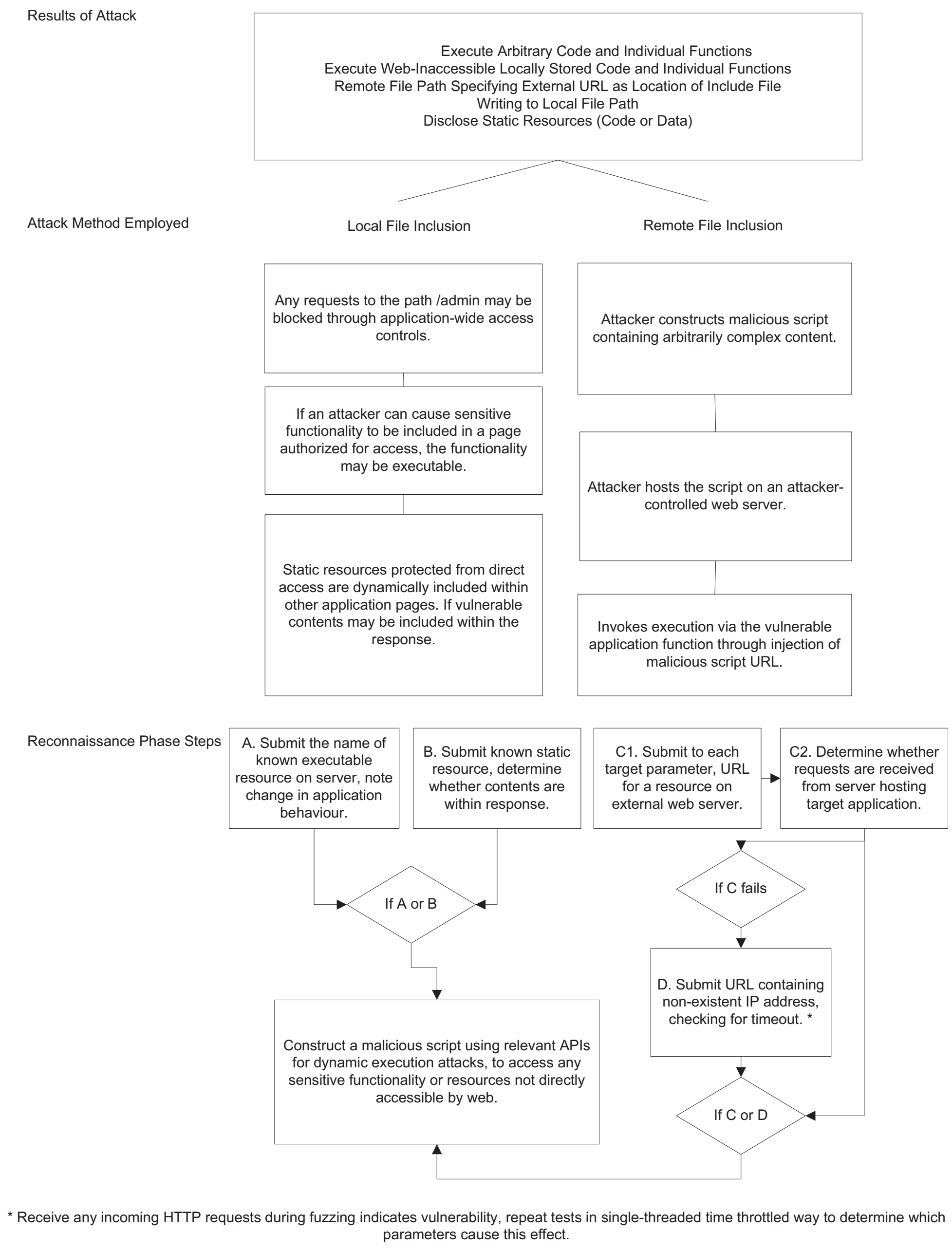

Results of Attack
Execute Arbitrary Code and Individual Functions
Execute Web-Inaccessible Locally Stored Code and Individual Functions
Remote File Path Specifying External URL as Location of Include File
Writing to Local File Path
Disclose Static Resources (Code or Data)
Attack Method Employed
Local File Inclusion
Remote File Inclusion
Any requests to the path /admin may be blocked through application-wide access controls.
If an attacker can cause sensitive functionality to be included in a page authorized for access, the functionality may be executable.
Static resources protected from direct access are dynamically included within other application pages. If vulnerable contents may be included within the response.
Attacker constructs malicious script containing arbitrarily complex content.
Attacker hosts the script on an attacker-controlled web server.
Invokes execution via the vulnerable application function through injection of malicious script URL.
Reconnaissance Phase Steps
A. Submit the name of known executable resource on server, note change in application behaviour.
B. Submit known static resource, determine whether contents are within response.
C1. Submit to each target parameter, URL for a resource on external web server.
C2. Determine whether requests are received from server hosting target application.
If A or B
If C fails
D. Submit URL containing non-existent IP address, checking for timeout. *
If C or D
Construct a malicious script using relevant APIs for dynamic execution attacks, to access any sensitive functionality or resources not directly accessible by web.
* Receive any incoming HTTP requests during fuzzing indicates vulnerability, repeat tests in single-threaded time throttled way to determine which parameters cause this effect.

Figure 8: Local or Remote File Inclusion Scenarios [360]

Regular Expression String ($Regex_{12}$):

```
(?:etc\/\W*passwd)
```

Possible Detected Attack String:

```
http://www.example.com/vulnerable.php?display=../../../../../../../../etc/passwd
%00
```

Description:

Generally, most inclusion attempts will not include a data file such as the `passwd` file. Instead, code execution that includes local files causes infintely more nefarious and devastating effects upon an intended target. This attack string will be caught by $Regex_{12}$, due to the fact that it contains the pattern `etc/passwd` This attack is difficult to detect due to the naming of a target file can be one of many default files, or a name that an attacker has contrived itself. An additional regular expression could be developed that checks for file inclusion where the path traversed terminates in the `/lib` section on a Unix system or the default `*.dll` folder on a Windows system. The inclusion attempt is often used in conjunction with a path traversal technique where one file uploaded by the first exploit and the second utilizes knowledge of that file and the location of executable code to achieve a malicious goal. The URL path traversal exhibits an attacker's ability to access more sensitive areas of the file system such as retrieving the contents of the `passwd` file on a Unix system.

Detects `firefoxurl://` Injections, Cache Poisoning and Local File Inclusion or Execution

Regular Expression String ($Regex_{28}$):

```
(?:firefoxurl:\w+\|)|(?:(?:file|res|telnet|nntp|news|mailto|
chrome)\s*:\s*[%&#xu\/]+)|(wyciwyg|firefoxurl\s*:\s*\/\s*\/)
```

Possible Detected Attack String:

```
<iframe src='firefoxurl://example.com" -chrome "javascript:alert(0)'></iframe>
```

Description:

The `(wyciwyg|firefoxurl\s*:\s*\/\s*\/)` section from $Regex_{28}$ specifies the grouping that captures the attack string. It is defined by searching for the appearance of the keyword `firefoxurl://` and allows some spaces between symbols.

The `firefoxurl://` vulnerability is made capable through an input validation flaw between IE and Mozilla Firefox. This security hole impacts users who have both browsers installed on their machine. Internet Explorer enables a user to specify arbitrary arguments to the process responsible for handling URL protocols. Upon installation on the Windows operating system, the program registers a URL protocol handler called `FirefoxURL`. The definitive entry location in the registry table is shown here:

```
[HKEY_CLASSES_ROOT\FirefoxURL\shell\open\command\@]
```

A sample key entry is included as follows:

```
C:\\PROGRA~1\\MOZILL~2\\FIREFOX.EXE -url "%1" -requestPending
```

When IE accepts a `firefoxurl://` styled link, it uses the above command to invoke Firefox, and renders the request using the associated URL of the following format:

```
FirefoxURL://address" -argument "script starts here
```

In our attack example, the `firefoxurl` is actually included in the `iframe` tag. More information about this type of attack could be found at [276, 248].

4.4 Cross Site Request Forgery

As outlined by the experts from the OWASP within the "Top 10 Application Security Risks of 2011" document [301], coming in at number five, are CSRF attacks. The attack itself exploits the target websites trust of a particular authenticated user. An attacker inserts and executes malicious requests from the perspective of a trusted authenticated user [329]. In more complex versions of this attack, the actions performed are completed in an automated fashion on the behalf of the validated user [302]. On the server side, the request from the victim appears legitimate. Therefore the vulnerable application responds with the requested data in a manner appropriate to a normally authenticated user. To check for this vulnerability on a web application, an investigator must validate if the parameters to any of the requests may be predetermined [360]. The following example demonstrates our forensic log tool to detect CSRF attempts based on URL, name or referrer payloads and cross domain requests mads in an attempt to retrieve JavaScript Object Notation (JSON) [124] data.

Figure 9 demonstrates scenarios where an attacker would employ CSRFs that attain given results. It also provides an outline for testing applications for CSRF vulnerabilities.

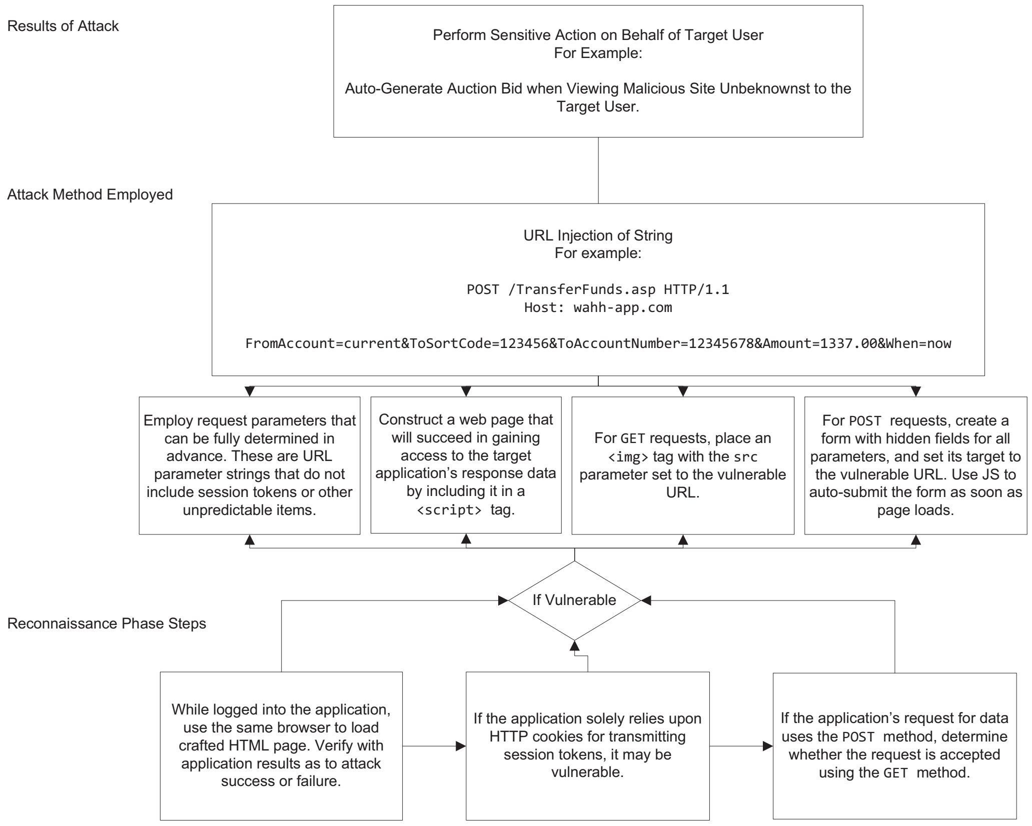

Results of Attack
Perform Sensitive Action on Behalf of Target User
For Example:
Auto-Generate Auction Bid when Viewing Malicious Site Unbeknownst to the Target User.
Attack Method Employed
URL Injection of String
For example:
POST /TransferFunds.asp HTTP/1.1
Host: wahh-app.com
FromAccount=current&ToSortCode=123456&ToAccountNumber=12345678&Amount=1337.00&When=now
Employ request parameters that can be fully determined in advance. These are URL parameter strings that do not include session tokens or other unpredictable items.
Construct a web page that will succeed in gaining access to the target application's response data by including it in a <script> tag.
For GET requests, place an <img> tag with the src parameter set to the vulnerable URL.
For POST requests, create a form with hidden fields for all parameters, and set its target to the vulnerable URL. Use JS to auto-submit the form as soon as page loads.
If Vulnerable
Reconnaissance Phase Steps
While logged into the application, use the same browser to load crafted HTML page. Verify with application results as to attack success or failure.
If the application solely relies upon HTTP cookies for transmitting session tokens, it may be vulnerable.
If the application's request for data uses the POST method, determine whether the request is accepted using the GET method.

Figure 9: Cross Site Request Forgery Scenarios [360]

4.4.1 Detects URL-, Name-, JSON, and Referrer-Contained Payloads

Regular Expression String ($Regex_4$):

```
(?:[+\/]\s*name[\W\d]*[)+])|(?:;\W*url\s*=)|
(?:[^\w\s\/?:>]\s*(?:location|referrer|name)\s*[^\/\w\s-])
```

Possible Detected Attack String:

```
<META HTTP-EQUIV="refresh" CONTENT="0;url=javascript:alert('XSS');">
```

Description:

This attack attempt borrowed from [334, 391], is detected by the grouping `(?:;\W*url\s*=)` in $Regex_4$. To provide an explanation of why this occurs, this group will match any string which contains a “;” followed by an indeterminate number of non-word characters, along with the keyword `url=`. It is important to note here that the match will still succeed if there exists an arbitrary number of whitespaces before the equal sign.

<`Meta`> is a tag used in both HTML/XHTML and usually resides in the head section of the page [307, 322]. The only operational dierence in usage of the tag between the languages is that the tag needs to be closed in XHTML, but not in HTML even though it provides similar functionality in both languages. In specific cases, this vulnerability may arise due to the fact that the tag is not closed. The <`Meta`> tag provides the definition of metadata regarding the current HTML document. Its content is not present on a rendering of the particular web page, but remains accessible for parsing. Typically this tag, specifies the page description including keywords, author of the document, and last modified time. This tag can be used to tell the browser when to display or reload a particular page or to define keywords for a search engine.

Among Meta tag attributes relevant to this attack is the `http-equiv` attribute, the keyword `refresh`, and the `content` attribute. The `http-equiv` attribute provides an HTTP header for the information in the `content` attribute. The keyword `refresh` indicates that the current page should be reloaded periodically. Finally, the `content` attribute could have a value “x” specifying the refreshing interval in seconds.

During an attack, should the `content` value be set to `content="5;url=http://example.com/"`, the current page will be automatically redirected to the specified location. Furthermore, if the `content` attribute is set to `0;url=javascript:alert('XSS');`, the script is executed in the browser upon rendering.

This particular <`Meta`> tag vulnerability applies to the browsers Firefox and Chrome. According to Mozilla Foundation Security Advisory (MFSA) 2009-22 [95], Mozilla has apparantly attempted to patch this vulnerability in Firefox. Currently the vulnerability allows refresh header to redirect to JS URIs. A way to bypass this redirection protection in Firefox is given in [285]. As such the payload of a successful attack script is encrypted using base64 [230] code as follows:

```
<meta http-equiv="refresh" content="0;url=data:text/html;base64,
PHNjcmlwdD5hbGVydChkb2N1bWVudC5jb29raWUpPC9zY3JpcHQ+">
```

When decoded into plaintext, the script appears as the following:

```
<script>alert(document.cookie)</script>
```

As with other injection techniques any bits of script code can potentially become embedded in the <`Meta`> tag for a variety of exploitational purposes.

4.5 Cross Site Scripting

Every application that is vulnerable to XSS is vulnerable to CSRF inherently, since once any code block can be executed, any other maliciously inserted web page may be automatically loaded [190]. XSS attacks are not limited to just HTML and JS. They encompasses a very broad definition of web application vulnerabilities since they represent an invaluable tool within a individuals aresenal that procures a diverse spectrum of outcomes. Thus, our current implementation of the log analysis tool does not encompass all of the techniques that can be employed to procure every possible XSS exploit. With the power and utility of the scripting language used, the numerous possible outcomes vary greatly and are dependant on the type of XSS that application is susceptible to. This attack occurs at any point when an attacker forces a web site to render malicious code to be executed in a user's web browser [190]. The intended victim is the end user; it does not attack the server directly but uses the site only as the conduit for the attack [190]. The next six subsections include samples of attacks and examples implemented in the tool that outline XML-based vulnerabilities, JS concatenation-based vulnerabilities, base encoding obfuscated XSS, two HTML-based vulnerabilities, and a Visual Basic Scripting edition (VBScript)-based [89] attack proving that other scripting languages can be used to mount XSS attacks.

XSS scenarios described in Figure 10 outline probing methods, and potential results of XSS attacks.

4.5.1 XML - Javascript DOM, Properties or Methods

Regular Expression String ($Regex_{15}$):

```
([^*:\s\w,.\/?+-]\s*)?(?<![a-z]\s)(?<![a-z\/_@>\-\|])(\s*return\s*)?
(?:create(?:element|attribute|textnode)|[a-z]+events?|setattribute|
getelement\w+|appendchild|createrange|createcontextualfragment|
removenode|parentnode|decodeuricomponent|\wettimeout|option|useragent)
(?(1)[^\w%"]|(?:\s*[^@\s\w%",.+\-]))
```

Possible Detected Attack String:

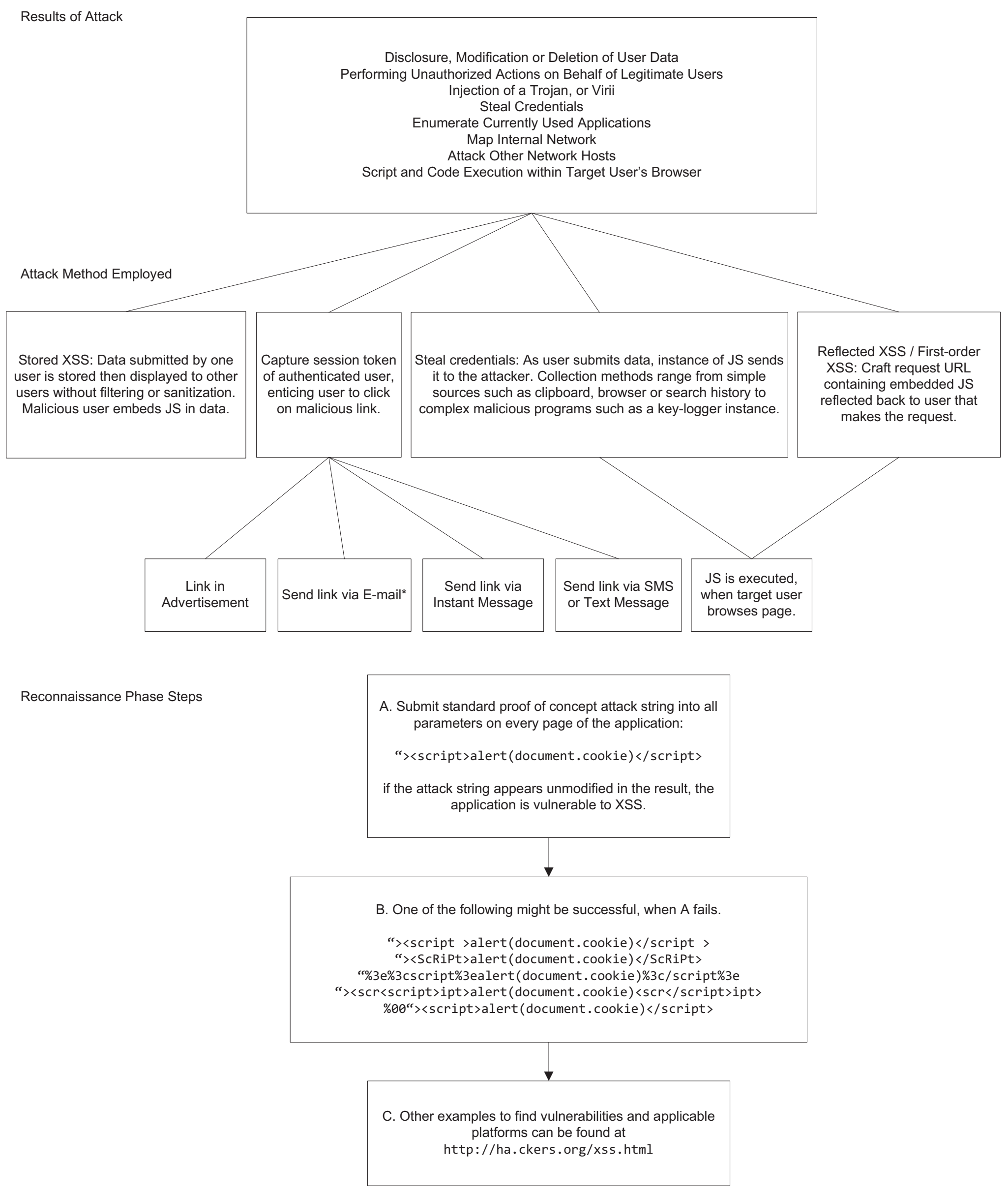
```
with(document)getElementsByTagName('head')[0].appendChild
(createElement('script')).src='http://www.attacker.com/att.js'
```

Description:

Matching of this attack string is presented in Table 14, some failed matching attempts in previous regular expression steps have been omitted from this table for brevity.

Table 14: Regular Expression Matching for $Regex_{15}$

Step	Regex	Input	Description
1	`[^*:\s\w,.\/?+-]`	(	after a few try, "(" is matched.
2	`\s*`	-	No space, successful match
3	`(?<![a-z]\s)`	-	peek behind, no letter or space found, successful match
4	`(?<![a-z\/_@>\-|])`	-	peek behind, no specified symbol found, successful match
5	`(\s*return\s*)?`	-	no given pattern found, successful match
6	`(?:create(?:element`	`createElement`	keyword found
7	`[^\w%"]`	(	since "(" is found in step 1, use `[^\w%"]` to match "(", successful match


Results of Attack
Disclosure, Modification or Deletion of User Data
Performing Unauthorized Actions on Behalf of Legitimate Users
Injection of a Trojan, or Virii
Steal Credentials
Enumerate Currently Used Applications
Map Internal Network
Attack Other Network Hosts
Script and Code Execution within Target User's Browser
Attack Method Employed
Stored XSS: Data submitted by one user is stored then displayed to other users without filtering or sanitization. Malicious user embeds JS in data.
Capture session token of authenticated user, enticing user to click on malicious link.
Steal credentials: As user submits data, instance of JS sends it to the attacker. Collection methods range from simple sources such as clipboard, browser or search history to complex malicious programs such as a key-logger instance.
Reflected XSS / First-order XSS: Craft request URL containing embedded JS reflected back to user that makes the request.
Link in Advertisement
Send link via E-mail*
Send link via Instant Message
Send link via SMS or Text Message
JS is executed, when target user browses page.
Reconnaissance Phase Steps
A. Submit standard proof of concept attack string into all parameters on every page of the application:
if the attack string appears unmodified in the result, the application is vulnerable to XSS.
B. One of the following might be successful, when A fails.
C. Other examples to find vulnerabilities and applicable platforms can be found at
http://ha.ckers.org/xss.html

```
"><script>alert(document.cookie)</script>
```

```
"><script >alert(document.cookie)</script >
"><ScRiPt>alert(document.cookie)</ScRiPt>
"%3e%3cscript%3ealert(document.cookie)%3c/script%3e
"><scr<script>ipt>alert(document.cookie)<scr</script>ipt>
%00"><script>alert(document.cookie)</script>
```

* To make the email appear to be legitimate, attacker uses images from the vulnerable site, or if the site has feedback functionality vulnerable to XSS, this may be used with the intended target's email address so that the message appears to be from the vulnerable company.

Figure 10: Cross Site Scripting Scenarios [360]

In this attack, the DOM method `createElement()` creates a <`script`> tag in the webpage, and imports scripts in the source file located at `http://www.attacker.com/att.js`. According to W3C [388], the XML DOM defines a standard way for accessing and manipulating XML documents. It presents an XML document as a tree-structure. According to the structure, web page elements can be added to the XML document as tree nodes, thus indicated by the name of the `createElement()` method. There exist a multitude of other methods that can be used to dynamically generate a web page, some of which cause this type of vulnerability. An interested reader can glean additional details from [107].

4.5.2 Javascript Concatenation Patterns

Regular Expression String ($Regex_{30}$):

```
(?:\+=\s*\(\s")|(?:!+\s*[\d.,]+\w?\d*\s*\?)|(?:=\s*\[s*\])|
(?:"\s*\+\s*")|(?:[^\s]\[\s*\d+\s*\]\s*[;+])|(?:"\s*[&|]+\s*")|
(?:\/\s*\?\s*")|(?:\/\s*\)\s*\[)|(?:\d\?.+:\d)|(?:]\s*\[\W*\w)|
(?:[^\s]\s*=\s*\/)
```

Possible Detected Attack String:

```
s1=’’+’java’+’’+’scr’+’’;s2=’’+’ipt’+’:’+’ale’+’’;
s3=’’+’rt’+’’+’(1)’+’’; u1=s1+s2+s3;URL=u1
```

Description:

One method of obfuscating injected JS code is to inject the variable in pieces as a part of multiple concatenation strings. As in this case, the malicious URL is composed of three different strings. The rendering engine joins these to read one string variable, and upon loading of the site, renders the malicious URL `javascript:alert(1)` along with the rest of the valid data on the page. The regular expression grouping that matches this string is `(?:"\s*\+\s*")`. Other methods of obfuscation are detailed in the next example using base encoding techniques.

4.5.3 Detects JavaScript Obfuscated by Base Encoding

Regular Expression String ($Regex_{25}$):

```
(?:=\s*[$\w]\s*[\(\[])|(?:\(\s*(?:this|top|window|self|parent|
_?content)\s*\))|(?:src\s*=s*(?:\w+:|\/\/))|(?:\w+\[("\w+"|\w+\|\|))|
(?:[\d\W]\|\|[\d\W]|\W=\w+,)|(?:\/\s*\+\s*[a-z"])|(?:=\s*\$[^([]*\()|
(?:=\s*\(\s*")
```

Possible Detected Attack String:

```
<IMG SRC=&#106;&#97;&#118;&#97;&#115;&#99;&#114;&#105;&#112;&#116;&#58;
&#97;&#108;&#101;&#114;&#116;&#40;&#39;&#88;&#83;&#83;&#39;&#41;>
```

Description:

This attack string is shown to be a match to $Regex_{25}$[1]. Note that the payload of this attack has been encoded in order to obfuscate desired code. In plain text, this script translates to `javascript:alert('XSS')`.

4.5.4 Detects HTML Breaking Injection

Regular Expression String ($Regex_1$):

```
(?:"[^"]*[^-]?>)|(?:[^\w\s]\s*\/>)|(?:>")
```

Possible Detected Attack String:

```
">
"  >
message=<script>alert("xss");</script>
```

Description:

This regular expression matches entries that include any string comprised of any combination of specified symbols with additional whitespace. Applications vulnerable to XSS attacks might return this input string directly to the user browser, as an example, it might be in the form of an error message. If any of these strings contains one or more injected scripts originating from an attacker, they will be executed by the client browser. Sometimes, HTML tolerates extra whitespaces in the

[1]http://demo.php-ids.org/

tag as they are automatically truncated. It is for this reason that any number of whitespaces are considered in the expression string.

This input contains the script `<script>alert("xss");</script>`, which causes the users browser to display a message box with the word "xss" on it. It is detected by the first non-capturing group `(?:"[^"]*[^-]?>)` of this `Regex`. The step-by-step matching process for this regular expression is presented in Table 15[2]:

Table 15: Regular Expression Matching For $Regex_1$

Step	Regex	Input	Description
1	`"`	`"`	Starting from the first symbol `"` in the input string
2	`[^"]*`	`xss`	Matching subsequent symbols
3	`[^-]?`	`"`	Matching subsequent symbol
4	`>`	`)`	Matching failed at `)`
5	`"`	`"`	Starting again from the second symbol `"`
6	`[^"]*[^-]?`	`);</script`	Matching subsequent symbols
7	`>`	`>`	Matching the last symbol, succeed

Analogously, instead of the `alert()` function, other malicious code can be injected as the payload for error messages. These scripts are capable of requesting vital information such as a session id from the user's browser. Generally, the more sensitive information is the most sought after by attackers to increasingly escalate the depth of an attack persuant to the desired outcome of the malicious intervention.

4.5.5 Attribute Breaking Injections

Regular Expression String ($Regex_2$):

```
(?:"+.*[<=]\s*"[^"]+")|(?:"\w+\s*=)|(?:>\w=\/)|
(?:#.+\)["\s]*>)|(?:"\s*(?:src|style|on\w+)\s*=\s*")|
(?:[^"]?"[,;\s]+\w*[\[\(])
```

Possible Detected Attack String:

```
var a="</script><script> alert('XSS !');</script><script>";
```

Description:

[2]Since the attack string matching process works exact the same for each expression in our knowledgebase, here we will not repeat this step-by-step illustration for other expressions.

This regular expression detects attribute breaking attacks under the condition that certain user request data with source URL contains the above string. For instance, consider this situation where an attacker directs a victim to a webpage embedded with a segment of code:

```
...
<script>
var a="</script><script> alert('XSS !');</script><script>";
...
</script>
...
```

Once this page is displayed by the victim's browser, it executes the embedded malicious script. As was the case with $Regex_1$, an attacker attains full access to the web application under the guise of the authorized user.

4.5.6 VBScript Injection

Regular Expression String ($Regex_{69}$):

```
(?:(?:msgbox|eval)\s*\+|(?:language\s*=\*vbscript))
```

Possible Detected Attack String:

```
<img src=1 language=vbscript onerror=msgbox+1>
```

Description:

This attack is captured by the first regular expression grouping specified by `(?:msgbox|eval)\s*\+`. It first looks for the keyword `msgbox` then detects a random number of whitespace characters, followed by the "+" sign. As a result, the final captured string appears as `msgbox+`.

VBScript is a Microsoft scripting language used as the default scripting language for ASP. Client--side VBScript only works in IE [107]. When a VBScript is inserted into an HTML document, the browser reads the HTML and executes the VBScript either immediately, or upon reciept of another queued event or user action.

The `MsgBox` function in the example displays a message box with the choice of two buttons, then waits for the user to click a button, and returns a value that indicates which button the user clicked. As this is just an innocuous example, other more malicious VBScript code could be injected instead by an attacker if the concerned web application is vulnerable to this attack vector.

In this attack, the VBScript is injected in the `img` tag, using the `src` attribute, but could very well be injected in other similarly oriented tags. Additional details may be perused at [348, 107].

4.6 Denial of Service

A DoS attack is classified by the outcome of a malicious action. Not all DoS attacks are generated through web application vulnerabilities, as this problem describes potential issues from an overall viewpoint of a networked system. When a successful DoS attack has commenced, the victim machine is forced to suspend service to legitimate users examples of which could be the result of the service being inundated with messages, the service being specifically shutdown or made unreachable through other avenues of attack [270]. Within this section two examples are provided as two varied methods of attack through web application vulnerabilities, one consisting of a SQL attack string, and the other consisting of an XSS attack string that both force a DoS against a target web application.

4.6.1 Detects MySQL Charset Switch and MSSQL DoS

Regular Expression String ($Regex_{52}$):

```
(?:alter\s*\w+.*character\s+set\s+\w+)|(";\s*waitfor\s+time\s+")|
(?:";.*:\s*goto)
```

Possible Detected Attack String:

```
ALTER TABLE 'users' CHANGE 'password' 'password' VARCHAR(255)
CHARACTER SET gbk COLLATE gbk_chinese_ci NOT NULL
```

Description:

According to $Regex_{52}$, the attack detection process is described in Table 16:

Table 16: Regular Expression Matching for $Regex_{52}$

Step	Regex	Input	Description
1	`alter\s*`	`ALTER`	Symbol sequence "`ALTER`"
2	`\w+`	`TABLE`	Keyword "TABLE"
3	`.*`	`‘users‘`	Matching any character
4	`.*`	`CHANGE`	Matching any character
5	`.*`	`‘password‘`	Matching any character
6	`.*`	`‘password‘`	Matching any character
7	`.*`	`VARCHAR(255)`	Matching any character
8	`character\s+`	`CHARACTER`	Keyword "CHARACTER "detected
9	`set\s+`	`SET`	Keyword "SET " detected
10	`\w+`	`gbk`	Keyword "gbk" detected

Altering the character set of a table or an individual column enables an attacker to evade certain functionality used by the database process to validate user input. For example, one PHP function vulnerable to multi-byte encoding is `addslashes`. A prerequisite for this attack specifies that the database must use a multi-byte charset, such as GBK Chinese [1] or BIG 5 [283]. The vulnerability is instantiated due to the fact that after modification of the charset, the PHP escaping facility is not aware of this modification. In fact, the validation function, which for our example was the `mysql_real_escape_string()`, still works on the basis of the default charset, which is by default the `latin1`.

Within the GBK character set, the character `0xbf27` is not a valid multi-byte character, but 0xbf5c is. Interpreted as single-byte characters, 0xbf27 is 0xbf (>) followed by 0x27 ('), and 0xbf5c is 0xbf (>) followed by 0x5c (\).

In real attack scenarios since it assumes that if an attacker attempts to set up an SQL injection against a MySQL database, which has single quotes escaped with a backslash, the regular avenues of attack fail. Unlike the scenario in which the database uses `addslashes()`, where the attacker would replace the injected character with `0xbf27`. This ensures that `addslashes()` modifies the injected data to become `0xbf5c27`, a valid multi-byte character followed by a single quote. Verily, the injection of a single quote is successful due to escaping mechanisms remaining maintained. This is applicable due to the fact that `0xbf5c` is interpreted as a single character, not two.

4.6.2 Detects Cross-Site Scripting Denial of Service

Regular Expression String ($Regex_{65}$):

```
(?:(^|\W)const\s+[\w\-]+\s*=)|(?:(?:do|for|while)\s*\([^;]+;+\))|
(?:(?:^|\W)on\w+\s*=[\w\W]*(?:on\w+|alert|eval|print|confirm|
prompt))|(?:groups=\d+\(\w+\))|(?:(.)\1{128,})
```

Possible Detected Attack String:

```
xyz onerror=alert(1);
```

Description:

The sub-string `onerror=alert` in this attack scenario is caught by the following regular expression grouping as indicated in Table 17:

```
(?:(?:^|\W)on\w+\s*=[\w\W]*(?:on\w+|alert|eval|print|confirm|prompt))
```

Table 17: Regular Expression Matching for $Regex_{65}$

Step	Regex	Input	Description
1	`(?:^|\W)`	`space`	Space character
2	`on\w+\s*`	`onerror`	Keyword `onerror`
3	`=[\w\W]*`	`=`	Symbol `=`
4	`alert`	`alert`	Keyword `alert`

This attack example explained in [53] shows that the code is injected between two script tags and fires an alert when executed. The `onerror` event is triggered when an error occurs during the loading of a document or an image. A typical use case of this event is: `onerror="SomeJavaScriptCode"` In the following example, an alert box will be displayed if an error occurs when loading an image:

```
<img src="image.gif" onerror="alert('The image could not be loaded.')">
```

4.7 Electronic Mail Spam

Spam has evolved from a message that originated from the Digital Equipment Corporation (DEC) [111] computer company as an annoucement of a version of their product over the ancient Advanced Research Projects Agency Network (ARPANET) [331] system in 1978 [368] to mass advertisements of deceptive, fraudulent or illegal material such as child pornography [310]. One method that changes the origin of unsolicited bulk electronic messages may be the product of an attacker's malicious action against a web application. Canada's Bill C-28 [286] specifies conditions upon which an electronic message is classified into this category, and more importantly prohibits the initiation of such a communication. The following sample from our tool includes a regular expression that detects mail header injections against vulnerable web applications.

4.7.1 Detect Common Mail Header Injection

Regular Expression String ($Regex_{63}$):

```
(?:[\w.-]+@[\w.-]+%(?:[01][\db-ce-f])+\w+:)
```

Attack String:

```
sender@anonymous.www%0ACc:rickastley@nevrgunna.xxx
%0ABcc:rick@roll.com,bacon@ch.ca
```

Description:

The attack example that can be detected by our regular expression is presented in detail at [240]. This reference also provides an excellent conceptual description of email header injection. Of the multitude of methods employed to send anonymous emails, exploitational outcomes that abuse such services provide the ability of mass electronic messaging or to spoof the identity and origin of the person utilizing the service. Implementations that provide anonymous mailing services are usually accessed through a web mail form, which executes the `mail()` function to generate electronic messages whose headers specify their origin as IP of the server providing the service. In this way the web application's mailing form acts as an SMTP proxy. It is common to specify a mail form that enables a potential user to control the contents of the subject, message, and the sender's email address fields. This can be implemented in the following way:

```
Function usage: mail([RECIPIENT],[SUBJECT],[MESSAGE],[EXTRAHEADERS],
[EXTRAPARAMS]); (mail())
```

Generally, extra parameters to the mail function are not supplied by the user, and the attack does not focus on these parameters, so these are excluded. However another avenue of attack may be present should the user be able to specify this field. Most webmasters hardcode the recipient's email address into the contact form of their web application. This is done in an attempt to eliminate this method of script exploitation. Unfortunately, this does not resolve the issue and a successful method of attack may be attempted when the function is instantiated as such:

```
<?php mail("rickastley@nevrgunna.xxx","A KABOOM","There was supposed to be
an earth shattering KABOOM!\n \n That creature has stolen the space
modulator!!","From: marvinthemartian@mars.xxx\n"); ?>
```

This ensures that the raw output appears similar to the following:

```
To: rickastley@nevrgunna.xxx
Subject: A KABOOM
From: marvinthemartian@mars.xxx


There was supposed to be an earth shattering KABOOM!


That creature has stolen the space modulator!!
```

Additional fields that can be specified in the mail headers are outlined in RFC 822 [104], of which examples include the Carbon Copy (CC) and the Blind Carbon Copy (BCC) fields. As specified in RFC 822, you must add a line feed for every header. The `LF` character has a hexadecimal value of `0x0A`. By providing the following values to the example script at [240]:

```
Sender: sender@anonymous.www%0ACc:rickastley@nevrgunna.xxx%0A
Bcc:rick@roll.com,bacon@ch.ca
Subject: ahem
Message: My Message...
```

The email's raw data is rendered this way:

```
To: rickastley@nevrgunna.xxx
Subject: ahem
From: sender@anonymous.xxx
Cc:recipient@someothersite.xxx
Bcc:rick@roll.com,bacon@ch.ca


My Message...
```

This shows that the mail headers were injected successfully, despite the fact that the only header value to be specified by the user from the HTML form is `From:`. This results in an email that is sent to three people of our choosing:
`rickastley@nevrgunna.xxx`, `somebloke@grrrr.xxx`, and `someotherbloke@oooops.xxx`.

In this example, both CC and BCC headers have been used to perform the injection. It would also have been possible to perform the injection by overloading the destination `To` header. In this case, the last value is added (as in the CC and BCC fields) to the hardcoded email address of the webmaster.

It is worth noting that the above $Regex_{63}$ does not detect this attack vector according to `[01][\db-ce-f]`. This is because it captures any hexadecimal value start with digit 0 or 1, except ones end with `a` or `d`. The regular expression renders the value `0x0A` to be safe as when specifying header information in an email, the end of each section terminates upon recognition of the `LF` character.

4.8 Conclusion

The subject of forensic log analysis includes many disparate parts. From the viewpoint of an intruder, the best way to break in is generally the front door. The front door to most servers is loosely defined as the port assigned to serve a web page as access to this resource is designed and maintained to be readily available for the purposes of communication. The matches that are detected in a log file provided for analysis outline a source of potential areas that should be investigated further. Attackers may probe other areas of the server for potential avenues of attack. With the discovery of potential attacks in this manner, perhaps similar efforts may procure additional results when comparing other data stores. Within this chapter, we aimed to provide examples of attacks, regular expressions designed to detect those attacks, and descriptions of how these matches are made by our forensic log analysis program. The next chapter we provide a description of our tool that we implemented in order to detect attack using regular expressions.

Chapter 5

Forensic Log Analysis Tool

This chapter details the design and implementation of our forensic log analysis and attack detection application. Attack detection is important in forensic perspective in order to procure an accurate time line. According to such a time line a successful attack can signal either the inception of active participation on behalf of a malicious force or a significant moment within a compounding scenario where an attempt has been made order to procure one of many ever escalating goals. This chapter includes an introduction to our tool with testaments to the tool's purpose and scope. We also include a description outlining considerations of the design and applicable project constraints. Then, we provide an outline of our method of implementation and a segment on the usability of our tool. The contributions of our tool provide advantages to a forensic investigator. In conclusion, an additional goal was discovered during the research of our tool. This goal shows that our implementation provides an initial design that forms the basis for additional modules. It is hoped that in the future a more encompassing application proving valuable to a forensic investigator can be implemented utilizing our methodology.

5.1 Purpose and Scope

The purpose of this project is to engineer a tool capable of forensically analyzing log files. During the development of this tool, the discovery of the key components vital to a forensic log analysis tool came to light. Beneficial to a forensic investigator are the ability to derive meanings from multiple events, the generation of a comprehensive timeline of events, the modification of existing or the ability to instantiate additional methods of defining relevant events, and the formatting of output representing this data in a manner that is easily read and understood, among others. Extending this tool to enable the import of a variety of log file types to our application would increase the scope and require further investigation into methods of relational mapping between varying types. Initially, a correlation option remains an invaluable and necessary component to the original design of the tool in order to generate meanings from seemingly innocuous but related events. The broad scope of an encompassing forensic tool forces us as designers to focus specifically on one particular avenue of log file and type of attack. Our tool is limited to those vectors of attack found within AWS `access_log` files, yet still reveals developmental details that can be expanded upon in future applications.

5.2 Design and Implementation

During the design phase several underlying principles were maintained. The first of which required that the tool must be easily adopted by end users. Thus, the user interface must maintain an intuitive design that follows usability principles. During an investigation, both expert witnesses or forensic investigators, who may be end users of our forensic tool, work within time constraints for each individual case. Therefore to perform the analysis the program must maintain a high-level of efficiency. Due to the limited scope, our tool is designed in a way that makes it both modular, which enables code reuse, and extensible in the sense that it maintains the ability to remould, repurpose or add match definitions in the future. Being written in Java, enables our tool to reach the widest user audience. This advantage of portability enables our tool to run on multiple platforms. Other forensic considerations we propose ensure applicability, integrity, and comprehensive reporting.

The implementation of our log analysis tool comply with the aforementioned design constraints. Indeed, the GUI design conforms to general guidelines [317] in order to associate familiarity with other applications. Efficiency is calculated as a measure of both organization and computational cost of the operations peformed within the application. Our method of organization includes employing an SQL database to store the regular expressions collection. This enables effective retrival of `Regexs` and the option to catalogue information about each one as they are added to the application. In order deploy the database, the first step must consist of parsing log file events. The use of the database facilitates effective selection and recall of data for correlation. The use of `Regexs` in our tool facilitates two of the design principles stated previously. In fact, they provide a low computational cost solution to find matches and the ability for the tool to remain configurable to those end users fluent in representing similar expressions. The `Regex` component comprises the central core of our program. It is the part of code that directly interacts with suspicious events recorded within analyzed log files. The effectiveness of attack detection within our application completely relies on the `Regex` knowledge base that is deployed. Finally, the choice of developing our application in the Java programming language ensures our tools modularity and its ability to run on multiple platforms. The application consists of the following five separate components whose purposes are log file processing, `Regex` processing, connecting to the database, attack detection, and finally the GUI. Figure 11 shows the process flow of our program.

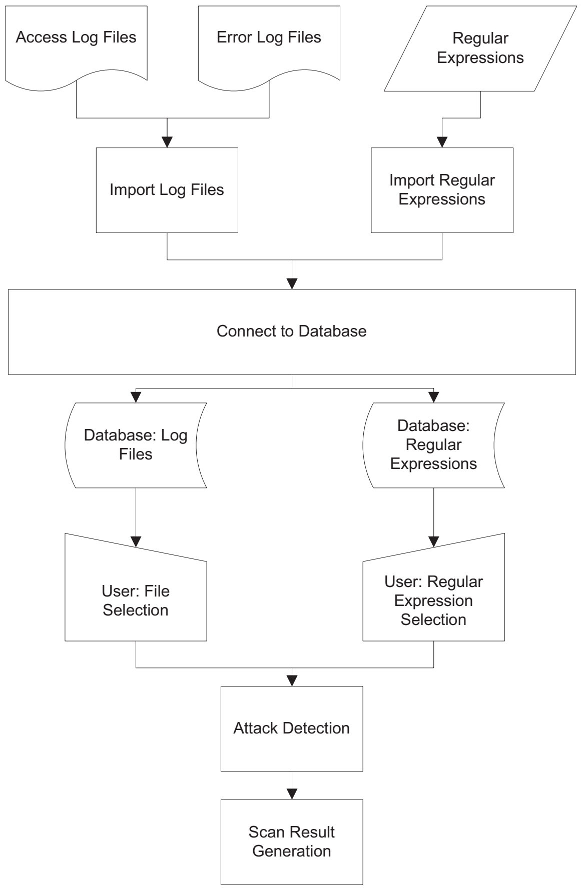


Figure 11: Program Flowchart

5.3 User Documentation

Web application security is a burgeoning subject of research within both the business and academic sectors concerned with cyber security. The development of methods to procure forensic trails which include significant events delves into every major network realm. The ability of our application to perform forensic log analysis on the AWS log files in the detection of web application attacks furthers the understanding and significance of these types of events, and how they relate to the attack scenarios or incident's timeline development. The functionality of the components that comprise our tool integrate in a synergistic manner to effectively and efficiently accomplish the requirements of its design. These functionalities include a user interface, interaction with a backend database, and the ability to parse, process and extract meaningful event data from target log files. Other than the base foundation for this application, our tool incorporates functionality which enables extensions of itself and encourages the development of fresh or the enhancement of existing attack definitions. This section is meant to incite understanding and incorporate a users perspective of how to effectively employ our application as a forensics log analysis tool. To explore how an end user navigates our application, we direct attention to four areas, which include environmental requirements for operation in Section 5.3.1, steps that prepare the tool to perform the analysis of a particular log file in Section 5.3.2, an outline for the definition of matching criteria in Section 5.3.3, and how to complete an execution for match detection in Section 5.3.4.

5.3.1 Operating Environment

Outlined below are the necessary prerequisites that must be provided in order to facilitate an environment appropriate to the functioning of our tool. We used the Java programming language in order to develop our forensic log analysis tool. Our implementation requires access to a database to retrieve regular expressions. During the testing and debugging phase, and now for demonstrational purposes, a MySQL 5.1 implementation is in use. This does not restrict end users from choosing their own SQL server instantiation, as our code functions independently from the genre of SQL server provided to it. That being said, our implementation requires that some type of SQL service be provided on the local machine our application is executed on. It is this service that is used to hold the `Regex` match defintions and the input log files.

Before launching our program, it is necessary that the SQL implementation is preconfigured to include a user account with the identifier of “root” and password equal to “root”. Our application uses this account for interactions with the MySQL database. During operation, a schema entitled ‘`log_analysis`’ is created to enable the storage of new tables, populate these tables with data and conduct operations on that data. Other preconfiguration necessary for our program to function correctly is the existence of several tables within the schema `log_analysis`. The three table types include one that provides storage for `Regex`s, one that details case information, and the last type stores the log file event data.

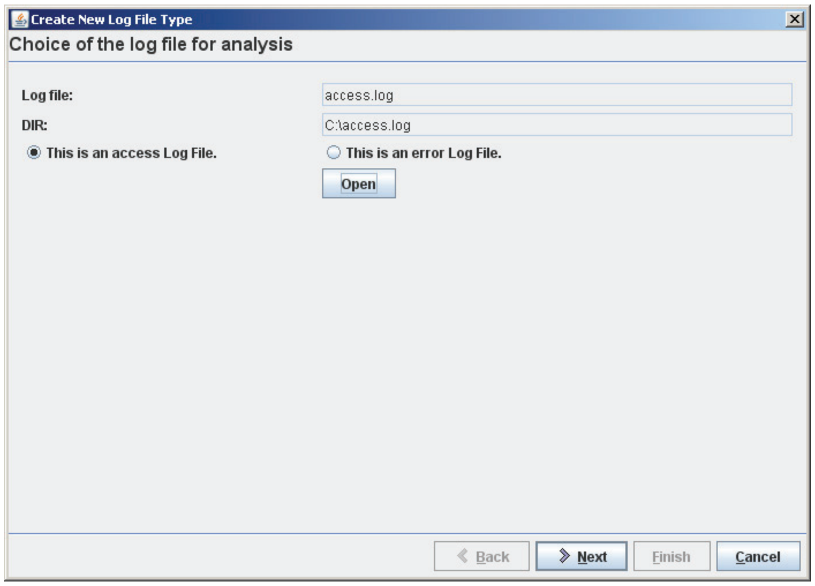
The `regexpr` table is used for the storage of both predefined and user defined `Regex`s that correspond to a number of attack types. There could be one or more `Regex`s that represents the same method of attack. These `Regex`s can be categorized into two subsets, “Access” and “Error”, which are applied to scan `access_log` files and `error_log` files, respectively. Event entry formats vary from one type of log file to the next. AWS access and error log files can sometimes utilize the same set of `Regex`s for particular attack scenarios. Those `Regex`s that cannot be applied to both file types are flagged within the database and filtered where the log file type is specified.

The `cases` table is used to store metadata information regarding target log files provided by the user as input. Every entry in this table represents a log file under investigation, and links and records the table name containing the stored content of this file. This removes unnecessary file I/O operations for those cases that analyze very large files that require repeated analysis.

The last type of table included within the schema are those labeled with the name of the imported log files. Each table of this kind is generated upon importation of the corresponding log file into the database. This data is stored until deletion is specified by the user. Each entry in this table corresponds and maintains the data of an event within the log file.

5.3.2 Analysis Preparation

Methods in accordance with the preliminary stages of the execution of this tool are described herein. Here we include a brief description of application launch, and describe how to import either AWS access or error log files into the database. After initialization, the main interface of our tool presents a menu bar containing the “Options” menu. The “Options” menu presents two possible choices: “New Case” menu item and “Existing Case” menu item. In order to add a new case to the database, the end user has to click on “Options - >New Case”. This starts a wizard dialog that guides the end user through the different steps of the pre-analysis process. As a first step, the user has to select the type of file to be subjected to the analysis, which could be either an `access_log` file or an `error_log` file. Each of these type of files triggers a specific processing path. Then, using the “Open File” dialog, the user selects the actual file to be imported into the database for analysis. After the file is selected, the filename and directory location are output to the “Log file:” and “DIR:” fields for confirmation (Fig. 12). Importation of the selected log file proceeds by clicking the button labelled “Next” to confirm the selection.



Log file: access.log

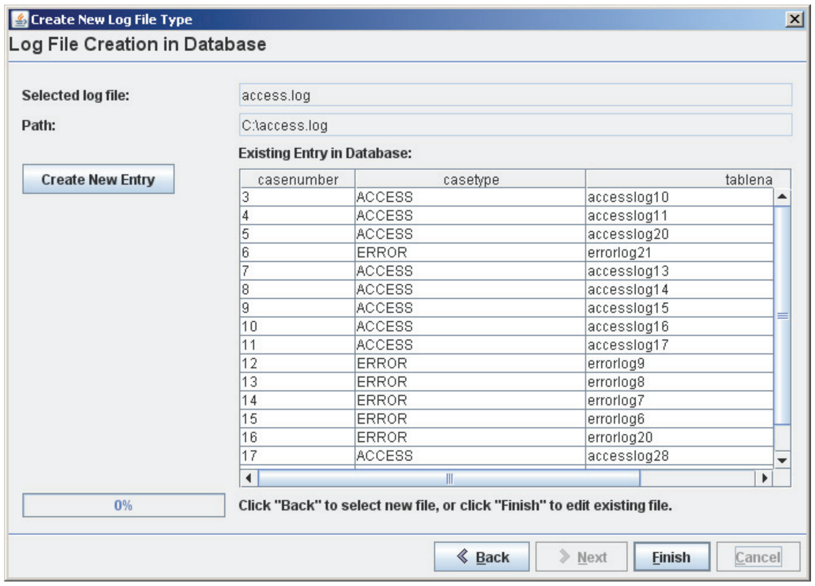
DIR: C:\access.log

(●) This is an access Log File. ( ) This is an error Log File.

Figure 12: Log Selection

Following the log file type declaration, program flow incites the user to click on the “Open” button which enables selection of the target log file. The window that provides access to the file system is is depicted in Figure 12.

In the next dialog shown in Fig. 13, the selected log file is imported into the database upon the click of the “Create new entry” button and its contents is displayed in the table on the right hand side of the window. Users can view the progress of the import operation as completion notification is made through the use of a progress bar at the bottom of the dialog. Once imported, the user can proceed to the next window in order to scan this file upon a click of the “Finish” button. At this step, the user is given the option to import another file or modify the current target file by clicking the “Back” button.



Log File Creation in Database

Selected log file: access.log

Path: C:\access.log

Existing Entry in Database:

casenumber	casetype	tablena
3	ACCESS	accesslog10
4	ACCESS	accesslog11
5	ACCESS	accesslog20
6	ERROR	errorlog21
7	ACCESS	accesslog13
8	ACCESS	accesslog14
9	ACCESS	accesslog15
10	ACCESS	accesslog16
11	ACCESS	accesslog17
12	ERROR	errorlog9
13	ERROR	errorlog8
14	ERROR	errorlog7
15	ERROR	errorlog6
16	ERROR	errorlog20
17	ACCESS	accesslog28

Click "Back" to select new file, or click "Finish" to edit existing file.

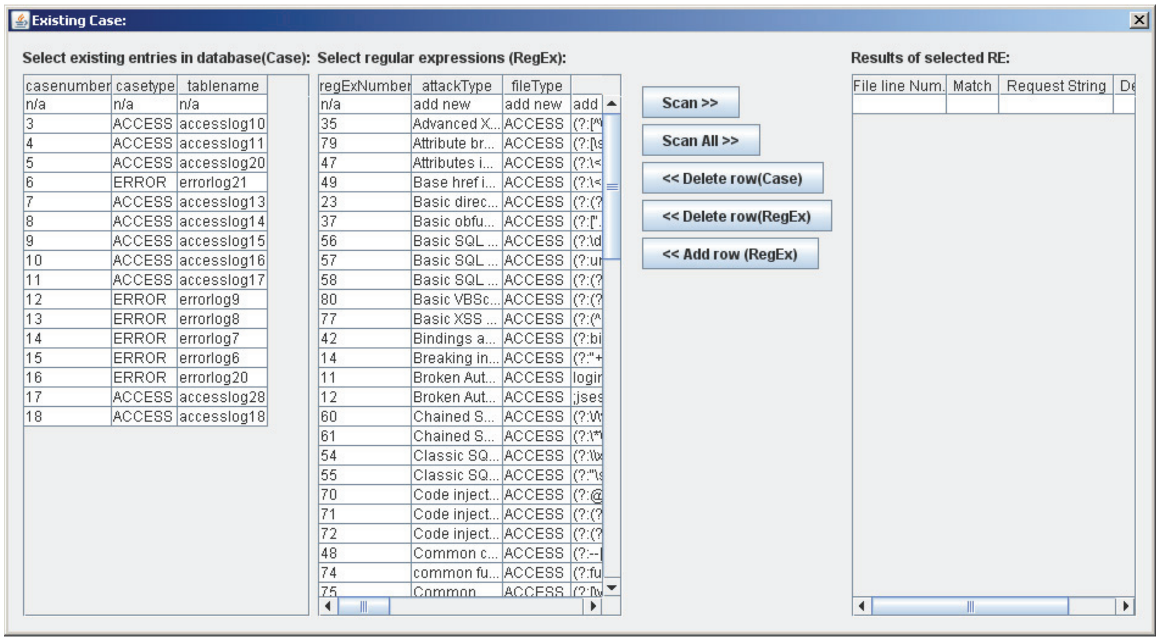
Figure 13: Log File Import

Upon clicking on the “Finish” button, the main interface is reloaded. At this point the user has the option to continue to scan imported log files or to include additional `Regex`s by choosing “Options → Existing Case”. Related operability issues are discussed within the next two in the following sections.

5.3.3 Match Definition

In this section, we guide the end user in the process of modifying the `Regex`s. Enriching and extending our knowledge base to identify new attacks assists end users in specific search terms and providing a faster method to perform repetative searches. The user interface of the modification dialog we showcase in Figure 14.

On the left hand side of this window resides a view of the `Case` table. This table contains metadata for all imported log files. Each row entry details the log file type (“case type”) and file name (“table name”) for imported log files. For each imported file, the filename is extracted and used to name the corresponding table within the SQL schema. It is this table that stores the entire content from the log file. For each log file case, an index number as indicated in the “case number” column is assigned. The user can not specify this value as it provides sequential numbers to those log files which are imported to tables. This panel also provides the capability for the user to delete unneeded imported log files from database using the “≪ Delete Row(Case)” button.



Select existing entries in database(Case):

casenumber	casetype	tablename
n/a	n/a	n/a
3	ACCESS	accesslog10
4	ACCESS	accesslog11
5	ACCESS	accesslog20
6	ERROR	errorlog21
7	ACCESS	accesslog13
8	ACCESS	accesslog14
9	ACCESS	accesslog15
10	ACCESS	accesslog16
11	ACCESS	accesslog17
12	ERROR	errorlog9
13	ERROR	errorlog8
14	ERROR	errorlog7
15	ERROR	errorlog6
16	ERROR	errorlog20
17	ACCESS	accesslog28
18	ACCESS	accesslog18

Select regular expressions (RegEx):

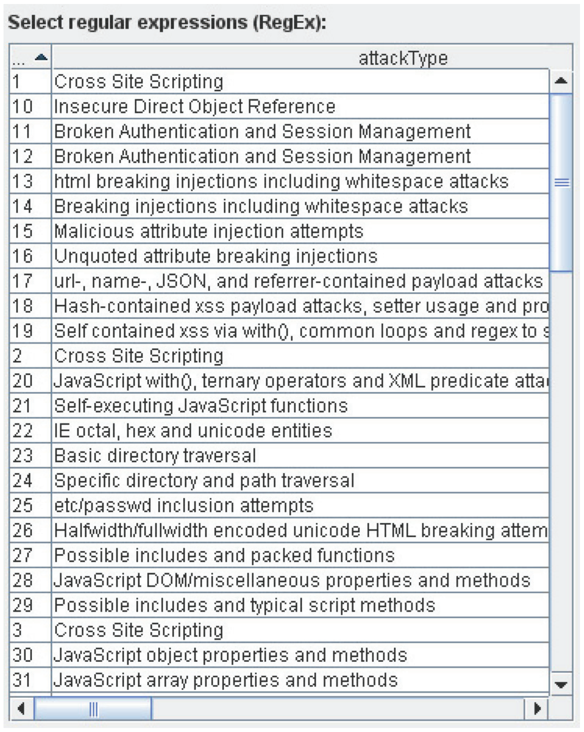
regExNumber	attackType	fileType	
n/a	add new	add new	add
35	Advanced X...	ACCESS	(?:[^
79	Attribute br...	ACCESS	(?:[\s
47	Attributes i...	ACCESS	(?:\<
49	Base href i...	ACCESS	(?:\<
23	Basic direc...	ACCESS	(?:(?
37	Basic obfu...	ACCESS	(?:[".
56	Basic SQL ...	ACCESS	(?:\d
57	Basic SQL ...	ACCESS	(?:ur
58	Basic SQL ...	ACCESS	(?:(?
80	Basic VBSc...	ACCESS	(?:(?
77	Basic XSS ...	ACCESS	(?:(^
42	Bindings a...	ACCESS	(?:bi
14	Breaking in...	ACCESS	(?:"+
11	Broken Aut...	ACCESS	logir
12	Broken Aut...	ACCESS	;jses
60	Chained S...	ACCESS	(?:\W
61	Chained S...	ACCESS	(?:*'
54	Classic SQ...	ACCESS	(?:\\x
55	Classic SQ...	ACCESS	(?:"\s
70	Code inject...	ACCESS	(?:@
71	Code inject...	ACCESS	(?:(?
72	Code inject...	ACCESS	(?:(?
48	Common c...	ACCESS	(?:--|
74	common fu...	ACCESS	(?:fu
75	Common...	ACCESS	(?:[\w

Results of selected RE:

File line Num.	Match	Request String	De

Figure 14: Regular Expression Selection, Scanning, and Results

In the center of the window, the content of the regular expression table, namely `regexpr`, is rendered. Each corresponding row entry includes an identifier, the type of attack string matched, the type of log file the `Regex` applies to and the `Regex` itself. The first column, “RegEx Number” represents an automatically assigned unique index generated for each `Regex`. The second column enables the end user to classify the type of attack that the corresponding `Regex` can match as illustrated in Figure 15. Certain types of attack have multiple avenues or payloads, and thus are interpreted into different `Regex`s. The unique name and number assists with classification and identification. The third column specifies the type of log file the `Regex` can be paired with for analysis purposes. Possible column values include “ACCESS”, “ERROR” or “ACCESS/ERROR”. The last selection “ACCESS/ERROR” is chosen if the expression is applicable for use in the analysis of either log file type. The fourth column stores the `Regex` employed to scan imported log files in order to perform attack detection analysis. Each of them was composed by extracting a set of features of an attack, which are log file events. The Figure. 16 provides a summary view of the available `Regex`s. Columns five and six contain values which specify respectively two optional conditions to be upheld during the analysis. The first contains a boolean value that, when set to "True", ignores letter case. The second, when set to "True", specifies ignorance of whitespaces within a matching event. Finally, the last column in this table provides a designated space for the user to add comments or descriptions for a specific `Regex` contained within the table.



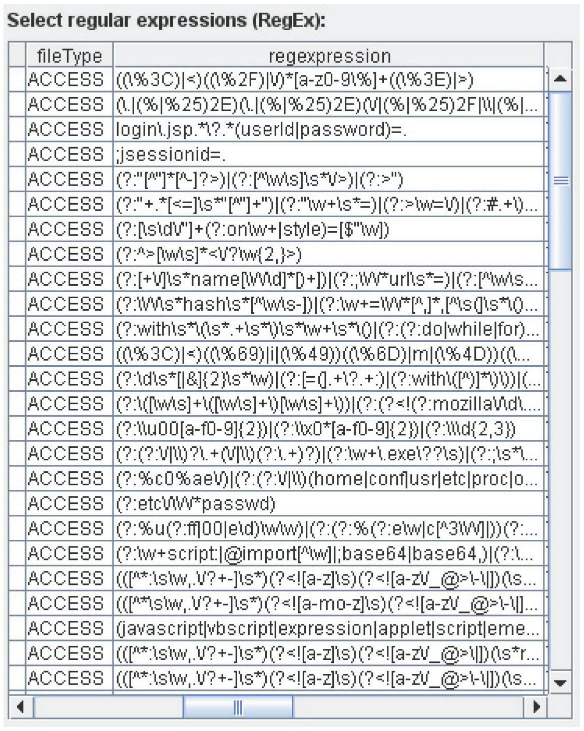
Select regular expressions (RegEx):

...	attackType
1	Cross Site Scripting
10	Insecure Direct Object Reference
11	Broken Authentication and Session Management
12	Broken Authentication and Session Management
13	html breaking injections including whitespace attacks
14	Breaking injections including whitespace attacks
15	Malicious attribute injection attempts
16	Unquoted attribute breaking injections
17	url-, name-, JSON, and referrer-contained payload attacks
18	Hash-contained xss payload attacks, setter usage and pro
19	Self contained xss via with(), common loops and regex to s
2	Cross Site Scripting
20	JavaScript with(), ternary operators and XML predicate atta
21	Self-executing JavaScript functions
22	IE octal, hex and unicode entities
23	Basic directory traversal
24	Specific directory and path traversal
25	etc/passwd inclusion attempts
26	Halfwidth/fullwidth encoded unicode HTML breaking attem
27	Possible includes and packed functions
28	JavaScript DOM/miscellaneous properties and methods
29	Possible includes and typical script methods
3	Cross Site Scripting
30	JavaScript object properties and methods
31	JavaScript array properties and methods

Figure 15: The Regular Expression Table I

Two buttons on this panel, "≪ Add row(RegEx)" and "≪ Delete row(RegEx)", are used to add or remove `Regex` entries from the database, respectively. When clicking on the "≪ Add row(RegEx)" button, the expression in the first editable row of this table is added to the analysis list. In order to retain the data entered, the user must confirm that all fields in this row, other than the index key, are specified before pressing this button. Otherwise the program determines it is an incomplete record and the data is not written to the table.

When the user clicks the "≪ Delete row(RegEx)" button, the data corresponding to the selected row is purged from the database.


Select regular expressions (RegEx):

fileType	regexpression
ACCESS	((\%3C)|<)((\%2F)|\/)*[a-z0-9\%]+((\%3E)|>)
ACCESS	(\.|(%|%25)2E)(\.|(%|%25)2E)(\/|(%|%25)2F|\\\|(%|...
ACCESS	login\.jsp.*\?.*(userId|password)=.
ACCESS	;jsessionid=.
ACCESS	(?:"[^"]*[^-]?>)|(?:[^\w\s]\s*\/>)|(?:>")
ACCESS	(?:"+.*[<=]\s*"[^"]+")|(?:"\w+\s*=)|(?:>\w=\/)|(?:#.+\)...
ACCESS	(?:[\s\d\/"]+(?:on\w+|style)=[$"\w])
ACCESS	(?:^>[\w\s]*<\/?\w{2,}>)
ACCESS	(?:[+\/]\s*name[\W\d]*[)+])|(?:;\W*url\s*=)|(?:[^\w\s...
ACCESS	(?:\W\s*hash\s*[^\w\s-])|(?:\w+=\W*[^,]*,[^\s(]\s*\(...
ACCESS	(?:with\s*\(\s*.+\s*\)\s*\w+\s*\()|(?:(?:do|while|for)...
ACCESS	((\%3C)|<)((\%69)|i|(\%49))((\%6D)|m|(\%4D))((\...
ACCESS	(?:\d\s*[|&]{2}\s*\w)|(?:[=(].+\?.+:)|(?:with\([^)]*\)\))|(...
ACCESS	(?:\([\w\s]+\([\w\s]+\)[\w\s]+\))|(?:(?<!(?:mozilla\/\d\....
ACCESS	(?:\\u00[a-f0-9]{2})|(?:\\x0*[a-f0-9]{2})|(?:\\\d{2,3})
ACCESS	(?:(?:\/|\\)?\.+(\/|\\)(?:\.+)?)|(?:\w+\.exe\??\s)|(?:;\s*\...
ACCESS	(?:%c0%ae\/)|(?:(?:\/|\\)(home|conf|usr|etc|proc|o...
ACCESS	(?:etc\/\W*passwd)
ACCESS	(?:%u(?:ff|00|e\d)\w\w)|(?:(?:%(?:e\w|c[^3\W]|))(?:...
ACCESS	(?:\w+script:|@import[^\w]|;base64|base64,)|(?:\...
ACCESS	(([^*:\s\w,.\/?+-]\s*)(?<![a-z]\s)(?<![a-z\/_@>\-\\\|])(\s...
ACCESS	(([^*\s\w,.\/?+-]\s*)(?<![a-mo-z]\s)(?<![a-z\/_@>\-\\\|]...
ACCESS	(javascript|vbscript|expression|applet|script|eme...
ACCESS	(([^*:\s\w,.\/?+-]\s*)(?<![a-z]\s)(?<![a-z\/_@>\\\|])(\s*r...
ACCESS	(([^*:\s\w,.\/?+-]\s*)(?<![a-z]\s)(?<![a-z\/_@>\-\\\|])(\s...

Figure 16: The Regular Expression Table II

5.3.4 Attack Detection

Our tool provides scanning capabilities against one or many types of attacks as shown in Figure 14. After the log file from the `Case` table is chosen, the "Scan ≫" option enables our tool to scan the log file against the attack type represented by the selected `Regex`s. The "Scan All ≫" option can be used to scan against all types of attacks available in the database to detect matches within the target log file. Scan results are displayed in the results table. Our tool does neither load the results into the database nor format them for a report style output. The results table includes entry data detailing matches. The "File line Num" column gives the line number of the target file in which the `Regex` matched an event. The `Match` field which takes a boolean value[1], describes whether the log entry in the current line matches any attack feature defined by its `Regex`.

For convenience and manual observation, our tool outputs the original HTTP requests in the "Request String" column. The last column specifies the attack type.

[1]We only display log entries that match at least one type of attack. In other words, only records have the "True" value in the `Match` field will be shown in the results table.

5.4 Non-Functional Requirements

Within this section we outline non-functional requirements that must be specified to provide sufficient data for relevance to an investigation when utilizing our tool for forensic analysis purposes. It is imperative the investigator knows those applications running on the server to enable the verification of the analysis results given by our tool and to enable search terms for other relevant information. Most importantly, in order to acquire an AWS `access_log` or `error_log` for analysis, the target machine must provide a running instance of the AWS. This will reveal other data sources or correlation information that may be resident on a machine under investigation. Our log tool does not provide definitive evidence of the success or failure of an attack attempt. This capability provides the interested audience an area for further investigation, for our implemention at this time, success must be determined through the use of additional analysis. According to forensic procedure, to ensure evidentiary value the integrity of the data must be preserved. Our program does not instantiate any file integrity processing capabilities so this must be ensured through another means. Each log file analyzed must be provided in raw and uncompressed format for input. Due to the volatile nature of log files, once an attacker has retained elevated priviledges, our program does not provide imaging or integrity verification for any changes that happen prior to analysis. Other than reading through the log file for the purposes of importation and searching, our `Regex`s do not modify the files contents.

5.5 Results

As was previously stated within the design and specifications, efficiency and the ability to process large file sizes are two top priorities of forensic investigators. To present an accurate description of regular expression efficency, our tool was tested with log files of varying sizes. Parsing and importing the log file is the most time intensive task of our forensic log analysis tool. For example, a 1.1 Gigabyte (Gb) log file was imported in 124624 milliseconds (ms) on an Intel Centrino Dual Core with 3 Gb of RAM and 6 Gb swap space. The processing time for testing a single regular expression against the same 1.1 Gb log file was 630 ms. Comparatively, a 209.2 kilobyte (kb) sized log file took 13200 ms to import and 40 ms for the same regular expression test.

5.6 Conclusion

In accordance with our design and specifications, we now present a tool that can assist in establishing an incident timeline. Significant events within the incident timeline are important areas to address during an investigation. Along with other forensic data, each originating attack matched with a regular expression from our tool can prove malicious intent. The events generated in the log file that match are anomalies crafted specifically for and only present in special circumstances. With the preceeding outline of our tool's purpose and scope, design and constraints, implementation method and usability considerations, we aim to provide sufficient information for future implementations that build on our development.

Chapter 6

Conclusion

6.1 Problem Statement

Present methods used to forensically analyse computer systems include tools such as Encase or the FTK. Although they feature prominently in case proceedings such as [102] or [223] they are both decidedly lacking in capabilities necessary to perform effective log analysis. There exists a need to engineer a forensic implementation capable of providing investigators the means to accomplish analysis whose results are efficient, and assist with reputable evidence generation. With regard to web application forensics, significant events recorded in the access log may be extracted that aide in the development of an incident timeline relevant to an investigation. Issues specific to the analysis of log files include the large size of target log files can be difficult for applications to handle efficiently, different web applications may or may not provide a means to log significant events, and as with most other log file types events that are recorded, those recorded by web applications and related services are cryptic in nature. It is the aim of this study to provide an implementation that forensically analyzes web applications which provides solutions to the aforementioned issues.

6.2 Solution

The program developed in accordance with our research findings which accompanies this document provides a legitimate web application forensic log analysis tool. Dealing with the massive scale of events typically recorded by applications, and compounding the number of events through comparison requires an innovative approach. Large file size and repeated access to similar data is made more efficent through the parsing and importing of such information into a database. For the source of our event files, some web applications provide their own logging mechanisms, however the web service itself by default provides a logging mechanism which records significant information relevant to an investigation. Specifically our solution looks to the AWS `access_log` file, as it is this file that records every request made to the web application's underlying web service itself. The implementation of this tool through the use of regular expressions ensures the accuracy, efficiency and relevance of the results provided to an investigator.

6.3 Contributions

One contribution of our study includes the development of a framework template which can be expanded, easily adapted and provide the basis for future forensic log analysis implementations. This study also details a comprehensive log tool comparision that leverages the current methods of performing log analysis in realms outside of forensic investigations. Our most important contribution details the collection of `Regex`s that detect attack injections against target web applications. These are organized into a variety of categories and descriptions are included which provide match definitions, attack examples and attack definitions. Research and development of the necessary elements a forensic log analysis tool are defined by contribute to the forensic capabilities of future applications.

6.4 Future Applications

During the course of our research several extensions that could be valuable to the development of an encompassing forensic log analysis tool are discovered. One such extension could be added to match same origin requests in order to timeline entire attack scenarios or group multiple related events. Other correlation sources determined by our research which may produce interesting event data include the output of modules such as `mod_session` or `mod_usertrack` which could include cookies or dynamic URL contents. As another extension to this type of correlation, within a network under forensic investigation, a proposed idea would be to provide not only event correlation for individual log file types, but cross-correlation between log file types for interpretation of events from different sources. This relates to the idea of performing event correlation on individual data sources first, and then attempting to designate the inferences made to generate new interpretations.

Other enhancement modifications include better report generation, development of new regular expressions to detect other attack scenarios, the ability to relate web application log events to the resulting access log events and the parsing or importation of different log file formats into similar modules. Invaluable to a forensic investigator would be the ability to monitor and analyze a variety of other log files for similar events including server cache log files or indirectly non-vulnerable web application specific log files residing on the target host. The ability to correlate log files collected from web servers to both other log files from that machine as well as log files from other servers on the same network can provide a more detailed and accurate timeline of events when attacks cross a network. Future applications should explore the extension or application of the definitions employed to find anomalies in network traffic to similar log event anomalies with regards to intrusion detection, protection systems and host integrity monitoring solutions.

6.5 Limitations

Our tool includes limitations that are set forth in this section. The first limitation we note is regarding the secure collection of data. This we deem beyond the scope of our parameters for the log tool we have developed and thus have not provided this functionality. Our application provides no assurances to the secure collection of the data that is being analyzed nor does it provide a method to detect tampering of the log file itself before, during or after processing. The application is also limited to only those attacks that may be performed on web applications, and restricted to those that may be recorded within the AWS `access_log` file. It applies only to those web servers that are serving web applications and attacks maybe attempted yet not successful due to the fact the server does not run the vulnerable web application applying to the particular attack that is detected.

Appendix A

Regular Expressions for SQL Injection

A.1 Detects MySQL Comments, Conditions, and ch(a)r Injections

Regular Expression String ($Regex_{40}$):

```
(?:"\s*(?:#|--|{))|(?:\/\*!\s?\d+)|(?:ch(?:a)?r\s*\(\s*\d)|
(?:(?:(n?and|x?or|not)\s+|\|\||\&\&)\s*\w+\()
```

Possible Detected Attack String:

```
SELECT * FROM users WHERE username = 'admin'--' AND password = 'foo'
```

Description:

The first group `(?:"\s*(?:#|--|{))`, matches upon recognition of one of these three distinct patterns, `"#`, `"--`, or `"{`. This definition also allows for any number of spaces between quote sign and the other symbols that succeed it.

The above attack example outlined in [360] is provided to illustrate an attack, which bypasses a login. The attack is based upon the principle that many applications which implement a forms-based login function use a database to store user credentials. For validation these applications perform a simple SQL query for each login attempt. The example command given below instructs the processor of the database to retrieve any record with username `rick` and password `secret`. Upon encountering such a record, the record is returned and the login process proceeds. If however, such a record does not exist in the database, the login process fails.

```
SELECT * FROM users WHERE username = ’rick’ AND password = ’secret’
```

An attack that can easily bypass this simple verification process is the scenario where the username `admin’--` is provided as input to the web application. The rendered SQL statement then becomes:

```
SELECT * FROM users WHERE username = ’admin’--’ AND password = ’foo’
```

Since the `--` denotes a comment in an SQL statement, whatever remains after the symbol sequence `--` will be regarded as such. The actual statement is then interpreted in such a way that bypasses the password verification altogether as long as there is a valid username which matches the input `admin`. Other alternative avenues of attack that produce similar effects also outlined within [360] such as:

```
SELECT * FROM users WHERE username = ’admin’
```

A.2 Classic SQL Injection Type 1

Regular Expression String ($Regex_{42}$):

```
(?:\\x(?:23|27|3d))|(?:^.?"$)|(?:^.*\\".+(?<!\\)")|
(?:(?:^["\\]*(?:[\d"]+|[^"]+"))+\s*(?:n?and|x?or|not|\|\||
\&\&)\s*[\d"[+&!@(),.-])|(?:[^\w\s]\w+\s*[|-]\s*"\s*\w)|
(?:@\w+\s+(and|or)\s*["\d]+)|(?:@[\w-]+\s(and|or)\s*[^\w\s])|
(?:[^\w\s:]\s*\d\W+[^\w\s]\s*".)
```

Possible Detected Attack String:

```
SELECT * FROM users WHERE username = ’’ OR 1=1--
```

Description:

Table 18: Regular Expression Matching for $Regex_{42}$

Step	Regex	Input	Description
1	`^["\\]*`	`\--`	No matched symbols found.
2	`[^"]+"`	`SELECT...username = ’`	Matched sub-string
3	`[\d"]+`	`’`	Single quote ’
4	`\s*`	`space`	Space “ ”
5	`(?:x?or)`	`or`	Keyword “or”
6	`\s*`	`space`	Space “ ”
7	`[\w"[+&!@(),.-]`	`1`	A digit `1`

Table 18 shows how $Regex_{42}$ matches this attack event. Considering the same login example as in $Regex_{40}$. Suppose that the attacker aims to discern the username of the administrator. In most applications, the first account in the database represents an administrative user. This is due to the fact that normally this account is created manually and then it is employed for the generation of all other accounts via application functionality. If the query returns the details for more than one user, in general, applications process the first user whose details are returned. An attacker can exploit this type of behavior to log in as the first user in the database by supplying the username followed by `’ OR 1=1--`. The outcome of such a successful attack returns the details of all application users as represented within the `users` table due to the logical construct `1=1`, which will always evaluate to true.

A.3 Classic SQL Injection Type 2

Regular Expression String ($Regex_{43}$):

```
(?:"\s*\*.+(?:or|id)\W*"\d)|(?:\^")|(?:^[\w\s"-]+(?<=and\s)(?<=or\s)
(?<=xor\s)(?<=nand\s)(?<=not\s)(?<=\|\|)(?<=\&\&)\w+\()|
(?:"[\s\d]*[^\w\s]+\W*\d\W*.*["\d])|(?:"\s*[^\w\s?]+\s*[^\w\s]+\s*")|
(?:"\s*[^\w\s]+\s*[\W\d].*(?:#|--))|(?:".*\*\s*\d)|
(?:"\s*or\s[\w-]+.*\d)|(?:[()*<>%+-][\w-]+[^\w\s]+"[^,])
```

Possible Detected Attack String:

```
SELECT * FROM users WHERE name = ’\’’; DROP TABLE users; --’;
```

Description:

The detection of this example is matched by `(?:"\s*[^\w\s?]+\s*[^\w\s]+\s*")` of this regular expression. The strings `'\'';` and `--'` within the event matches as shown in the following Table 19.

Table 19: Regular Expression Matching for $Regex_{43}$

Step	Regex	Input	Description
1	`"`	'	Single quote '
2	`\s*[^\w\s?]+`	\	\ symbol
3	`\s*[^\w\s]+`	'	' symbol
4	`\s*"`	'	Single quote '
5	`\s*[^\w\s?]+`	-	- symbol
6	`\s*[^\w\s]+`	-	- symbol
7	`\s*"`	'	Single quote '

The attack scenario as shown above clears all data within the table `users`. The severity of the consequences of such an attack range from the removal of all authentication mechanisms so that they default to open access, to a similar attack targetting a different table that could cause loss of valuable proprietary data contained within the database. Another regular expression $Regex_{48}$ presented in A.8 also matches the `DROP` term, however this example we include here for clarity. The reader is reminded that several attack scenarios may be detected through different regular expressions contained within our collection.

A.4 SQL Authentication Bypass Types 1, 2, 3

Regular Expression String ($Regex_{44}$):

```
(?:\d"\s+"\s+\d)|(?:^admin\s*"|(\/\*)+"+\s?(?:--|#|\/\*|{)?)|

(?:"\s*or[\w\s-]+\s*[+<>=(),-]\s*[\d"])|(?:"\s*[^\w\s]?=\s*")|

(?:"\W*[+=]+\W*")|(?:"\s*[!=|][\d\s!=+-]+.*["(].*$)|(?:"\s*[!=|]

[\d\s!=]+.*\d+$)|(?:"\s*like\W+[\w"(])|(?:\sis\s*0\W)|

(?:where\s[\s\w\.,-]+\s=)|(?:"[<>~]+")
```

Regular Expression String ($Regex_{45}$):

```
(?:union\s*(?:all|distinct|[(!@]*)?\s*[([]\s*select)|

(?:\w+\s+like\s+\")|(?:like\s*"\%)|(?:"\s*like\W*["\d])|

(?:"\s*(?:n?and|x?or|not|\|\||\&\&)\s+[\s\w]+=\s*\w+\s*having)|
(?:"\s*\*\s*\w+\W+")|(?:"\s*[^?\w\s=.,;)(]+\s*[(@"]*\s*\w+\W+\w)|
(?:select\s*[\[\]()\s\w\.,-]+from)
```

Regular Expression String ($Regex_{46}$):

```
(?:(?:n?and|x?or|not |\|\||\&\&)\s+[\s\w+]+(?:regexp\s*\(|sounds\s+
like\s*"|[=\d]+x))|("\s*\d\s*(?:--|#))|(?:"[%&<>^=]+\d\s*(=|or))|
(?:"\W+[\w+-]+\s*=\s*\d\W+")|(?:"\s*is\s*\d.+"?\w)|
(?:"\|?[\w-]{3,}[^\w\s.,]+")|(?:"\s*is\s*[\d.]+\s*\W.*")
```

Possible Detected Attack String:

```
UPDATE users SET password='newsecret' WHERE user = 'admin' or 1=1
```

Description:

During regular expression matching, the first pattern that the grouping `(?:where\s[\s\w\.,-]+\s=)` attempts to match is the keyword `where`. A successful match requires this pattern to be succeeded by a space character then one or more of the following symbols as shown in this list `[\s\w\.,-]`. Finally, the `=` symbol preceded by a space character concludes the successful capture match. As a result, the symbol sequence `WHERE user =` is captured from the given attack string example.

This attack, appears similar in nature to the case of $Regex_{42}$ shown in A.2. The difference is that this attacker intends to modify user credentials in the `users` table, instead of retrieving their values. Since this attack does not discern any specific user, the successful attack resets the value of every entry in the password column, which in turn resets the password for all of the application's users.

Table 20 exhibits the matching process for the $Regex_{45}$. The table after it, Table 21, explains how the $Regex_{46}$ matches the same attack string event in a different way.

Table 20: Regular Expression Matching for $Regex_{45}$

Step	Regex	Input	Description
1	`"\s*`	’	Single quote ’
2	`[^?\w\s=.,;)(]+`	`--`	`--` symbol
3	`\s*`	`space`	Space character
4	`[(@"]*\s*`	’	Symbol sequence “’ ”
5	`\w+`	`AND`	Keyword “AND”
6	`\W+`	`space`	Non-word character: space “”
7	`\w`	`p`	A letter `p`

Table 21: Regular Expression Matching for $Regex_{46}$

Step	Regex	Input	Description
1	`"`	’	Single quote ’
2	`|?`	-	No symbol matched
3	`[\w-]{3,}`	`admin`	Keyword “admin” detected
4	`[^\w\s.,]+`	’`--`	symbol sequence “’`--`”
5	`"`	’	Single quote ’

A.5 Concatenated SQL Injection and SQLLFI

Regular Expression String ($Regex_{47}$):

```
(?:^\s*[;>"]\s*(?:union|select|create|rename|truncate|load|
alter|delete|update|insert|desc))|(?:(?:select|create|rename|
truncate|load|alter|delete|update|insert|desc)\s+(?:concat|
char|load_file)\s?\(?)|(?:end\s*\);)|("\s+regexp\W)
```

Possible Detected Attack String:

```
’ UNION SELECT username,password,uid FROM users--
```

Description:

With regards to $Regex_{47}$, the first group looks for the start-of-a-line character specified by `^`, this is succeeded by an indeterminate amount of whitespace, and then by any symbol in the list `;>"`. After this the keyword `union` preceeded by an indeterminate amount of whitespace.

This attack makes use of the UNION operator which is used in SQL to combine the results of two or more SELECT statements into a single result set. When a web application contains a SQL injection vulnerability that occurs in a SELECT statement, an attacker can often employ the UNION operator to perform a second separate query, and combine the results of both queries. If the results of the query are returned to your browser, then this technique can be used to easily extract arbitrary data from within the database.

An applicable example describes a bookstore application, which uses the following statement to look for book information specified by the user to search for books by the associated publisher:

```
SELECT author,title,year FROM books WHERE publisher = 'Wiley'
```

A skillful malicious attacker instead provides a crafted publisher name similar to the above attack string. Upon reciept of this input, the application attaches the selection operation to the original request. In this case, all usernames and passwords from the `users` table will be returned in addition to the book information. It is also possible for this second query to extract data from a different database table and combine it with results from the first operation to return any data within the entire SQL schema.

When the results of two queries are combined using the UNION operator, the two result sets must have the same structure. This entails that they must contain the same number of columns of compatible data types, and each column is specified in the same order. In order to inject a second query that will return interesting results, the attacker must know the name of the database table and the names of the columns that contain the target data[360].

The SQL local file inclusion match that is performed by this regular expression is denoted by all of the groupings where the keywords `load`, `rename` and `load_file` are explicitly matched. For a further explanation of local file inclusion, we refer the interested reader to Section 4.3.4.

A.6 Chained SQL Injection Type 1

Regular Expression String ($Regex_{48}$):

```
(?:\/\w+;?\s+(?:having|and|or|select))|(?:\d\s+group\s+by.+\()|
(?:(?:;|#|--)\s*(?:drop|alter))|(?:(?:;|#|--)\s*(?:update|
insert)\s*\w{2,})|(?:[^\w]SET\s*@\w+)|(?:(?:n?and|x?or|not |
\|\||\&\&)\s+\w+[!=+]+[\s\d]*["=(])
```

Possible Detected Attack String:

```
'; insert into users values( 666, 'attacker', 'foobar', 0xffff )--
```

Description:

In this case, the engine first uses `(?:;|#|--)` to match the symbol ";", followed by a space, then the pattern matches the keyword `insert` using `(?:update|insert)`. The remaining part (`\s*\w{2,}`) then specifies a match of the string " `into`" .

Recall the previous example in $Regex_{42}$ where the attacker obtained all usernames and passwords from the `users` table. Here an attacker attempts to add a new user record using the `insert` operation. In particular, this vulnerability occurs where the database service is offered by a MS-SQL implementation. This is due to the fact that the MS-SQL service inherently allows multiple disparate SQL queries, separated by the semicolon character, to be batched together and run as if processing a single query. Moreover, the MS-SQL implementation does not require the two statements to be in any way related, therefore permitting an even wider attack range for this type of injection. [360]

A.7 Chained SQL Injection Type 2

Regular Expression String ($Regex_{49}$):

```
(?:\*\/from)|(?:\+\s*\d+\s*\+\s*@)|(?:\w"\s*(?:[-+=|@]+\s*)+[\d(])|
(?:coalesce\s*\(|@@\w+\s*[^\w\s])|(?:\W!+"\w)|(?:";\s*(?:if|while|
begin))|(?:"[\s\d]+=\s*\d)|(?:order\s+by\s+if\w*\s*\()
```

Possible Detected Attack String:

```
' or 1 in (select @@version)--
```

Description:

The detection process of the above attack string is matched in accordance with the `Regex` grouping `(?:coalesce\s*\(|@@\w+\s*[^\w\s])`. First, the two `@@` symbols immediately followed by the keyword `version` successfully satisfy the grouping.

As mentioned in [360], this attack technique can be used to extract arbitrary data from an Open DataBase Connectivity (ODBC) database. This is made possible through the generation of error messages that contain the value of the string object when an ODBC database attempts to cast an item of string data to a numeric data type, but the cast is unsuccessful. If error messages are being returned to the browser, they could provide a conduit for data to leak back to an attacker disclosing arbitrary stored data and vital system information. With this scenario it is possible to inject into the `WHERE` clause of a `SELECT` statement in order to perform a secondary query that triggers a resulting failed string conversion. The example presented above provides such an instantiation that returns this version information about the database and operating system:

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting
the nvarchar value 'Microsoft SQL Server 2000 - 8.00.194 (Intel X86)
Aug 6 2000 00:57:48 Copyright (c) 1988-2000 Microsoft Corporation
Enterprise Edition on Windows NT 5.0 (Build 2195: Service Pack 2) '
to a column of data type int.
```

Further exploitation of this vulnerability could retrieve other arbitrarily stored data. Perhaps a likely target would consist of passwords from a specified `users` table. In the following example, the password "0wned" of the user "admin" is specified and returned within an error message as shown below.

```
' or 1 in (select password from users where username='admin')--
```

```
Microsoft OLE DB Provider for ODBC Drivers error '80040e07'
[Microsoft][ODBC SQL Server Driver][SQL Server]Syntax error converting
the varchar value '0wned' to a column of data type int.
```

A.8 SQL `benchmark`, `sleep` Injection with Conditional Queries

Regular Expression String ($Regex_{50}$):

```
(?:(select|;)\s+(?:benchmark|if|sleep)\s?\(\s?\(?\s?\w+)
```

Possible Detected Attack String:

```
select if(user() like 'root@%', benchmark(50000,sha1('test')), 'false')
```

Description:

As shown in Table 22, the $Regex_{50}$ captures the attack string `select if(user`.

Table 22: Regular Expression Matching for $Regex_{50}$

Step	Regex	Input	Description
1	`(select|;)`	`select`	Keyword `select`
2	`\s+`	`space`	Space character " "
3	`(?:benchmark|if|sleep)`	`if`	Keyword "if" detected
4	`\s?\(`	`(`	Symbol `(` detected
5	`\s?\(?\s?`	`-`	No symbol detected
6	`\w+`	`user`	Keyword "user" detected

In MySQL, the `benchmark` function's use is to perform a specified action repeatedly. Instructing the database to perform a processor-intensive action, such as a hash function, a large number of times will result in a measurable time delay. In some cases, where the web server seemingly does not return any valuable information to the attacker through ordinary means, an attacker may attempt to use methods of causing time delays. The time delay of the web server can be interpreted as one particular indication or response. If such a delay command is executed, and the delay is instantiated, then the attacker infers the boolean value of a particular condition. Even if the content of the application's response is identical in the two cases, the presence or absence of the time delay enables the attacker to extract a single bit of information from the database. By performing numerous such queries, the attacker can systematically retrieve arbitrarily complex data from the database, one bit at a time.

A.9 MySQL UDF or Data/Structure Manipulation

Regular Expression String ($Regex_{51}$):

```
(?:create\s+function\s+\w+\s+returns)|(?:;\s*(?:select|create|
rename|truncate|load|alter|delete|update|insert|desc)\s*[\[(]?\w{2,})
```

Possible Detected Attack String:

```
\'; DESC users; --
```

Description:

As shown in Table 23, the $Regex_{51}$ specifies that an attack string containing "`; desc users`" is detectable.

Table 23: Regular Expression Matching for $Regex_{51}$

Step	Regex	Input	Description
1	`;\s*`	`;`	Symbol sequence "`; `"
2	`desc`	`desc`	Keyword "desc"
3	`\s*`	`space`	Space character detected
4	`[\[(]?`	`-`	No symbol detected
5	`\w{2,}`	`users`	Table name "users" detected

The SQL plus `desc` command returns the value of the column names and their respective data types. An injection of this type leaks valuable data most sought after during an attacker's reconnaisance phase of the attack scenario against a particular target.

The `(?:create\s+function\s+\w+\s+returns)` grouping of this regular expression matches an attack injection that creates or calls a user defined function. User defined functions are capable of providing access to underlying operating system commands and can be instantiated when the target system is running a MySQL database service. [176]

A.10 MySQL and PostgreSQL Stored Procedure Calls

Regular Expression String ($Regex_{53}$):

```
(?:procedure\s+analyse\s*\()|(?:;\s*(declare|open)\s+[\w-]+)|
(?:create\s+(procedure|function)\s*\w+\s*\(\s*\)\s*-)|
(?:declare[^\w]+[@#]\s*\w+)|(exec\s*\(\s*@)
```

Possible Detected Attack String:

```
exec(@variable)
```

Description:

The attack detection process for the above string is described in Table 24.

Table 24: Regular Expression Matching for $Regex_{53}$

Step	Regex	Input	Description
1	`exec`	`exec`	Keyword `exec`
2	`\s*`	-	No character matched
3	`\(`	`(`	Matching `(` symbol
4	`\s*`	-	No character matched
5	`@`	`@`	Matching `@` symbol

The `exec` operation is supported by all MS-SQL implementations to enable the dynamic execution of statements. One example of dynamic execution passes a string representation of a particular statement to the function. A side effect of this feature allows a malicious user to employ string manipulation techniques as any part of the statement in order to bypass filters designed to block certain expressions. Consider the following example:

```
declare @q varchar(8000)
select @q = 0x73656c656374202a2066726f6d207573657273
exec(@q)
```

The statement string is created from a hex-encoded numeric data. After decryption this evaluates to "`select * from users`". This command is then passed to the `exec` function and bypass many kinds of input filter. One such filter could block single quotation marks, yet the service still remains vulnerable due to attacks such as the one described beforehand. In an Oracle database, a similar statement defined by the keywords `EXECUTE IMMEDIATE` also executes queries that are input as encoded strings.

A.11 Postgres `pg_sleep`, `waitfor` Delays and DB Service Shutdowns

Regular Expression String ($Regex_{54}$):

```
(?:select\s*pg_sleep)|(?:waitfor\s*delay\s?"+\s?\d)|
(?:;\s*shutdown\s*(?:;|--|#|\/\*|{))
```

Detected Attack String:

```
; shutdown--
```

Description:

According to the regular expression grouping `(?:;\s*shutdown\s*(?:;|--|#|\/\*|{))`, the entire given attack string matches. During processing, the regular expression first matches events that include the ";" symbol, the keyword `shutdown` and the comment symbol `--` in this order of submission. Attacks that target the data itself include information leakage, sensitive information disclosure, manipulation or deletion. When the goal of an attacker is not data oriented, a denial of service attack causes potentially more damage should the main source of income for a potential target be procured from the availability of the service. A web application that relies on a database backend can immediately be crippled by the input of only 12 characters. The command that turns off an MS-SQL database is `shutdown`.

A.12 Match AGAINST, MERGE, EXECUTE IMMEDIATE and HAVING

Regular Expression String ($Regex_{56}$):

```
(?:merge.*using\s*\()|(execute\s*immediate\s*")|
(?:\W+\d*\s+having\s+\d)|(?:match\s*[\w(),+-]+\s*against\s*\()
```

Possible Detected Attack String:

```
' group by users.ID, users.username, users.password,
users.privs having 1=1--
```

Description:

As can be observed from the grouping `(?:\W+\d*\s*having\s*[^\s])`, this regular expression detects a sequence containing at least one non-word character followed by the keyword `having` and an amount of whitespace, and then at least one character. In this example, the substring " `having 1`" is captured.

This attack example is provided in [360] as an illustration of how to enumerate table and column names, without authorization from the web application. The resulting message discloses the name of the column after the specified column given in the query. Enumeration of all columns from a target table is possible when multiple queries are formed where the results from each query are input to each subsequent query.

A.13 MySQL Comment or Space-Obfuscated Attack

Regular Expression String ($Regex_{57}$):

```
(?:select\s*\*\s*from)|((?:select|create|rename|truncate|load|alter|
delete|update|insert|desc)\s*\(\s*space\s*\()
```

Possible Detected Attack String:

```
SELECT * FROM users WHERE username = '
```

Description:

The regular expression grouping denoted by `(?:select\s*\*\s*from)` detects the substring `SELECT * FROM` from the attack input. As one of the simplest SQL statements provided as input, this attack queries the indicated username from the `users` table. Based on the return values of this input, an attacker can determine the validity of each username. This accomplished, other attack techniques could employ the resultant data or futher exploitation through an additional query when an attacker employs the `union` operation.

Appendix B

Regular Expressions for Operating System Command Injection

B.1 MSSQL Code Execution and Reconnaisance

Regular Expression String ($Regex_{55}$):

```
(?:from\s+information_schema\W)|(?:(?:(?:current_)?user|database|
schema|connection_id)\s*\([^\)]*)|(?:";?\s*(?:select|union|having)
\s*["(\d])|(?:\wiif\s*\()|(?:exec\s+master\.)|(?:union select@)|
(?:union[\w(\s]*select)|(?:select.*\w?user\()|
(?:into[\s+]+(?:dump|out)file\s*")
```

Possible Detected Attack String:

```
'; exec master..xp_cmdshell 'ipconfig > test.txt' --
```

Description:

The substring `exec master.` signifies an attack of this type and is detected by the following grouping `(?:exec\s+master\.)` of the regular expression $Regex_{55}$. This attack makes use of the `exec` operation to enable the execution of an operating system command. In this injection, the malicious user is attempting to invoke a command shell on a MS Windows system, then utilizing elevated priviledges of the service, the shell calls the networking configuration command and outputs this to a text file. During the reconnaisance phase of an attack scenario, a malicious force attempts to procure sensitive data about the target system. The network configuration of a target system is an invaluable source of data that when mapped may reveal other attack avenues.

Appendix C

Regular Expressions for Cross Site Request Forgery

C.1 Data: URL Injections, VBS Injections and Common URI Schemes

Regular Expression String ($Regex_{27}$):

```
(?:data:.*,)|(?:\w+\s*=\W*(?!https?)\w+:)|(jar:\w+:)|
(=\s*"?\s*vbs(?:ript)?:)|(language\s*=\s?"?\s*vbs(?:ript)?)|
on\w+\s*=\*\w+\-"?
```

Possible Detected Attack String:

```
<a href="data:text/html;charset=utf-8,%3cscript%3ealert(1);
history.back();%3c/script%3e">XSS Example</a>
```

Description:

There are two grouping's specified in $Regex_{27}$ responsible for reporting the successful event whose details match the specified attack string. Each of these two groupings `(?:data:.*,)` and `(?:\w+\s*=\W*(?!https?)\w+:)` return successfully when the following substrings match `data:text/html;charset=utf-8,` and `href="data:`. Our example attacker's method of injection is to place the script code within the link tag's (`<a>`) `href` property. As a method to avoid detection the `<script>` tag is encoded as `%3cscript%3e`. An injection of the JS function `history.back()` redirects the user that clicks on the malicious link generated by the injection back to the previous page.

Appendix D

Regular Expressions for XSS

D.1 Detects Basic XSS probings

Regular Expression String ($Regex_{21}$):

```
(?:,\s*(?:alert|showmodaldialog|eval)\s*,)|(?::\s*eval\s*[^\s])|
([^:\s\w,.\/?+-]\s*)?(?<![a-z\/_@])(\s*return\s*)?(?:(?:document\s*\.)?
(?:.+\/)?(?:alert|eval|msgbox|showmodaldialog|prompt|write(?:ln)?|
confirm|dialog|open))\s*(?(1)[^\w]|(?:\s*[^\s\w,.@\/+-]))|
(?:java[\s\/]*\.[\s\/]*lang)|(?:\w\s*=\s*new\s+\w+)|
(?:&\s*\w+\s*\)[^,])|(?:\+[\W\d]*new\s+\w+[\W\d]*\+)|(?:document\.\w)
```

Detected Attack String 1:

```
<SCRIPT>document.write("<SCRI");</SCRIPT>PT
SRC="http://ha.ckers.org/xss.js"></SCRIPT>
```

Detected Attack String 2:

```
<A HREF="javascript:document.location='http://www.google.com/'">XSS</A>
```

Description:

Both of the above input strings classify as attacks when the regular expression grouping `(?:document\.\w)` processes from them the keyword `document.` followed by any word character.

The first detected attack string is injected and lays dormant until the browser renders the injected material, in this case the page loads the specified script file `xss.js`. This script file is hosted remotely from the web application, and using JS instructs the page to write the string `This is remote text via xss.js located at ha.ckers.org` and then prints out the values of all cookies available to the webpage [386]. The second detected attack string describes a potentially malicious redirection link definition.

D.2 JavaScript Cookie Stealing and Redirection

Regular Expression String ($Regex_{26}$):

```
(?:[^:\s\w]+\s*[^\w\/](href|protocol|host|hostname|pathname|hash|port|
cookie)[^\w])
```

Possible Detected Attack String:

```
<?xml-stylesheet type="text/css" href="style.css"?>
```

Description:

During processing of the regular expression $Regex_{26}$, the substring `" href=` identifies this particular string as an attack. This pattern denotes any character other than whitespace characters, to match the double quote symbol within the attack string. The pattern then looks for any number of whitespaces, including none, followed by any character not a whitespace which in our example equates to a space `" "` character. Finally, the keyword `href` matches to denote a positive result. The intended use for the stylesheet tag was to provide a conduit for including style preferences for the layout of XML documents specified in CSS. CSS define modifications to the style and layout of particular pages, which specify their use [108]. However, in this attack the malicious user exploits the behaviour of this tag to inject a CSS file that contains script.

D.3 Hash Redirection XSS, Set or Get Usage, Property Overloading

Regular Expression String ($Regex_5$):

```
(?:\W\s*hash\s*[^\w\s-])|(?:\w+=\W*[^,]*,[^\s(]\s*\()|(?:\?"[^\s"]":)|
(?:(?<!\/)__[a-z]+__)|(?:(?:^|[\s)\]\}])(?:s|g)etter\s*=)
```

Possible Detected Attack String:

```
"><script>eval(location.hash.substr(1))</script><!--
```

Description:

The grouping `(?:\W\s*hash\s*[^\w\s-])` of $Regex_5$ matches this attack string in the following manner. This grouping defines a match based on the succession that includes a non-word character, followed by arbitrary number of whitespace characters, followed by the keyword "`hash`, then followed by another arbitrary number of whitespace characaters, and any character that does not fall into the categories of being a word character, whitespace character, or a "`-`". In our attack string the sequence `.hash.` matches.

The attack string exemplifies a redirection attack, which uses the JS `location.hash()` function. This intended use of this function is to redirect the browser to specific HTML assigned anchor points on a page. The `location` object contains the current main URL content and the `hash()` object provides functions that modify or retrieve the anchor string. Within the URL, the hash sign (`#`) denotes the beginning of the anchor name. To give an example, this current URL describes a scenario where the `location.hash` value is equivalent to the string `section2`.

```
http://www.example.com/index.html#section2
```

The anchor function provides attackers with an URL specified means of executing code. Enabling direct execution for XSS attacks even when the allowance for input includes only a small number of characters, or special characters and symbols exluded as input for the application, may restrict an attacker to input directly into the URL itself.

Consider the following URL:

```
http://www.example.com/xss/attack.html?attk="><script>
eval(location.hash.substr(1))</script><!--#[attackscript]
```

The script portion `location.hash.substr(1)` evaluates the anchor data provided after the `#` sign. Should this portion be contain malicious scripts, these are executed by the browser.

This attack exhibits two distinct advantages. The first advantage is that the attack itself leaves very small little data with which to concoct a footprint that distinguishes its malicious nature from regular requests. This is due to the fact that the script segment, which occurs after the `#` sign, is not transmitted to the server. It is executed on the client side. The second advantage to this variety of injection is that it allows the attacker to bypass any limitation specified for the user provided string. Plenty of web applications restrict the total number of characters that a user can input for this reason. However, as shown in the above URL, the length of attack script is only 53 characters long:

```
"><script>eval(location.hash.substr(1))</script><!--
```

Bibliography

[1] Information Technology Universal Multiple-Octet Coded Character Set Part 1: Architecture and Basic Multilingual Plane. http://www.yys.ac.cn/gfbz/scanning/zfjhxxbm/gfbz26.htm&usg=ALkJrhhf9kbQcKKrQ5heN5m5H7AWlb6gqQ, December 1993. Last Accessed: April 20, 2011 at 11:54 EST.

[2] Tina Bird Abe Singer. *Building a Logging Infrastructure*, volume 12. SAGE - Short Topics in Systems Administration, 2560 Ninth Street, Suite 215, Berkeley, CA USA 94710, 2004. ISBN: 1-931971-25-0.

[3] LLC. AccessData Group. Forensic Toolkit (FTK) Computer Forensics Software | AccessData. http://accessdata.com/products/forensic-investigation/ftk, January 2010. Last Accessed: April 17, 2011 at 12:50 EST.

[4] Peter Adams. Features - Open Web Analytics Wiki. http://wiki.openwebanalytics.com/index.php?title=Features, December 2010. Last Accessed on January 6, 2011 13:20 EST.

[5] Peter Adams. Open Web Analytics - Main Page. http://www.openwebanalytics.com/, December 2010. Last Accessed on January 6, 2011 12:36 EST.

[6] Peter Adams. Welcome to Open Web Analytics - The Open Source Web Analytics Framework. http://wiki.openwebanalytics.com/index.php?title=Main_Page, December 2010. Last Accessed on January 6, 2011 12:46 EST.

[7] Adobe. *PDF Reference*, volume 1.7. Adobe Systems Inc., 7930 Jones Branch Drive, Fifth Floor McLean, VA 22102, 6th edition, November 2006. Last Accessed: April 11, 2011 at 21:24 EST.

[8] Amit Agarwal. The Total Number of Websites on Earth. http://www.labnol.org/internet/blogging/the-total-number-of-websites-on-earth/2257/. Written February 2008.

[9] agiardullo, borthakur, davefet, and facebook_bobby. Scribe. http://sourceforge.net/projects/scribeserver/, October 2009. Last Accessed: April 7, 2011 at 9:52 EST.

[10] Jakob Ahlin. Intrusion Detection C Part III - Web Application Firewalls Securing Web Applications with Mod_Security. http://193.11.99.44/PageFiles/32078/IDS-Modsecurity.pdf, January 2011. Last Accessed March 13, 2011, 10:01 AM.

[11] Alfred Aho, Peter Weinberger, Brian Kernighan, Paul Rubin, and Jay Fenlanson. *GAWK(1) Utility Commands - Man Page*. Free Software Foundation, Boston, MA, 1st edition, October 2007.

[12] Julie D. Allen, Deborah Anderson, Joe Becker, Richard Cook, Mark Davis, Peter Edberg, Asmus Freytag, Richard Ishida, John H. Jenkins, Rick McGowan, Lisa Moore, Eric Muller, Addison Phillips, Michel Suignard, and Ken Whistler. Unicode 6.0.0. http://www.unicode.org/versions/Unicode6.0.0/, 1991-2011. Last Accessed: April 7, 2011 at 20:50 EST.

[13] Robbie Allen. *Active Directory Cookbook*. O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 3rd ed. edition, December 2008. Ch. 4.

[14] Andres Andreu. *Professional Penetration Testing for Web Applications*. Wiley, Wiley Publishing, Inc. 10475 Crosspoint Boulevard Indianapolis, IN 46256, 2006.

[15] Anonymous. *Maximum Apache Security*. SAMS, Indianapolis, Indiana, USA 46240, June 2002.

[16] ANSI. American National Standards Institute - ANSI Homepage. http://www.ansi.org/. Last Accessed: April 7, 2011 at 20:30 EST.

[17] Vidur Apparao, Steve Byrne, Mike Champion, Scott Isaacs, Arnaud Le Hors, Gavin Nicol, Jonathan Robie, Peter Sharpe, Bill Smith, Jared Sorensen, Robert Sutor, Ray Whitmer, and Chris Wilson. Document Object Model (DOM) Level 1 Specification. http://www.w3.org/TR/DOM-Level-1/, October 1998. Last Accessed: April 5, 2011 at 11:24 EST.

[18] Opera Software ASA. Opera Browser | Faster & Safer Internet | Free Download. http://www.opera.com/, January 2011. Last Accessed: April 4, 2011 at 14:53 EST.

[19] Web Analytics Association. Web Analytics Association - About Us. http://www.webanalyticsassociation.org/?page=aboutus, January 1998. Last Accessed on December 28, 2010 13:31 EST.

[20] Todd Atkins. *README of swatch-3.2.3*. Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA, 3.2.3 edition, June 2008. Last Accessed on January 6, 2011 16:02 EST.

[21] Todd Atkins. Simple Log Watcher. http://sourceforge.net/projects/swatch/, June 2008. Last Accessed on January 6, 2011 16:02 EST.

[22] Creative Commons Attribution. Comparison of Web Browsers. http://en.wikipedia.org/wiki/Comparison_of_web_browsers, 09 2010.

[23] Jeff Atwood. Coding Horror: Microsoft LogParser. http://www.codinghorror.com/blog/2005/08/microsoft-logparser.html, August 2005. Last Accessed on January 5, 2011 17:47 EST.

[24] Robert Auger. Brute Force. http://projects.webappsec.org/w/page/13246915/Brute-Force, December 2009. Last Accessed: April 4, 2011 at 20:01 EST.

[25] Charles Aulds. *Linux Apache Web Server Administration*. Sybex, Inc., Alameda, California, USA 94501, 2001.

[26] Pablo Neira Ayuso. `netfilter` / `iptables` Project Homepage - The netfilter.org Project. http://www.netfilter.org/, January 1999-2010. Last Accessed: April 4, 2011 at 18:54 EST.

[27] Hans-Joachim Baader. *README for `acctsum` 1.0*. Free Software Foundation, Boston, MA, 1.5 edition, February 1998.

[28] Ryan C. Barnett. *Preventing Web Attacks with Apache*. Addison Wesley Professional, Pearson Education, Inc., Upper Saddle River, NJ, 2006. Ch. 5 (Mod_Security Section).

[29] Bradford L. Barrett. Home of the Webalizer. http://www.webalizer.org/, November 2009. Last Accessed January 6, 2011, 11:15 EST.

[30] Bradford L. Barrett. *The Webalizer - A Web Server Log File Analysis Tool README*. Free Software Foundation, Inc.,, 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA, 2.23-03 edition, October 2010. E-mail: brad@mrunix.net Last Download Accessed on: December 20, 2010 at 17:36 EST.

[31] Michael D. Bauer. *Linux Server Security*. O'Reilly Media, O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, 2nd edition edition, January 2005. ISBN: 0-596-00670-5.

[32] Cord Beermann. Calamaris Home Page. http://cord.de/tools/squid/calamaris/, March 2006. Last Accessed on December 28, 2010, 14:05 EST.

[33] David I. Bell. *scanSquidLog Man Page*. NEC, http://canb.auug.org.au/, 1.1 edition, March 2000. email: dbell@canb.auug.org.au.

[34] Lisa Bogar. SUID, SGID and Fix-modes. http://www.homepage.montana.edu/~unixuser/051602/SUID.html, May 2002. Last Accessed: April 16, 2011 at 22:57 EST.

[35] Paolo Bonzini. *SED(1) User Commands - Man Page*. Free Software Foundation, Boston, MA, 1 edition, June 2009.

[36] Bert Bos, Tantek elik, Ian Hickson, and ″kon Wium Lie. Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. http://www.w3.org/TR/2010/WD-CSS2-20101207/, Decemeber 2010. Last Accessed: April 5, 2011 at 11:15 EST.

[37] Inc. Boutell.com. Linux Software Map: `acctsum`. http://www.boutell.com/lsm/lsmbyid.cgi/002025, February 1998. Last Accessed on December 24, 2010 at 18:32 EST.

[38] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, and Franois Yergeau. Extensible Markup Language (XML) 1.0. http://www.w3.org/TR/xml/, November 2008. Last Accessed: April 5, 2011 at 12:27 EST.

[39] Bryan Burns, Jennifer Stisa Granick, Steve Manzuik, Paul Guersch, Dave Killion, Nicolas Beauchesne, Eric Moret, Julien Sobrier, Michael Lynn, Eric Markham, Chris Iezzoni, and Philippe Biondi. *Security Power Tools*. O'Reilly Media, OReilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472, first edition edition, August 2007.

[40] Steven Campbell. How Does Facebook Work? The Nuts and Bolts [Technology Explained]. http://www.makeuseof.com/tag/facebook-work-nuts-bolts-technology-explained/, February 2010. Page Last Accessed on: November 10, 2010 14:02 EST.

[41] Rich Cannings. Flash Authoring Tools Create Flash Files That Contain Cross-Site Scripting Vulnerabilities. http://www.kb.cert.org/vuls/id/249337, 06 2008. Google Security Team author is a member of.

[42] Brian Carrier. The Sleuth Kit (TSK) & Autopsy: Open Source Digital Investigation Tools. http://www.sleuthkit.org/, January 2003-2011. Last Accessed: April 17, 2011 at 16:07 EST.

[43] E. Casey. *Digital Evidence and Computer Crime: Forensic Science, Computers and the Internet*. Academic Press, 2nd edition, 2004. Ch. 6 & 22.

[44] A Caudwell. Logstalgia - Project Hosting on Google Code. http://code.google.com/p/logstalgia/, March 2010. Last Accessed on January 3 23:52 EST.

[45] Vinton G. Cerf and Robert E. Kahn. A Protocol for Packet Network Intercommunication. *IEEE Transactions on Communications*, 22(5):637–648, May 1974.

[46] Austin Cheney. Shutting Down XSS with Content Security Policy. http://blog.mozilla.com/security/2009/06/19/shutting-down-xss-with-content-security-policy/, 09 2010.

[47] James Clark and Steve DeRose. XML Path Language (XPath). http://www.w3.org/TR/xpath/, November 1999. Last Accessed: April 16, 2011 at 19:19 EST.

[48] Justin Clarke, Rodrigo Marcos Alvarez, Dave Hartley, Joseph Hemler, Alexander Kornbrust, Haroon Meer, Gary OLeary-Steele, Alberto Revelli, Marco Slaviero, and Dafydd Stuttard. *SQL Injection Attacks and Defense*. Syngress, Syngress Publishing, Inc. Elsevier, Inc. 30 Corporate Drive Burlington, MA 01803, January 2009.

[49] CNN.com. Teen Claims Responsibility for Disrupting Twitter. http://www.cnn.com/2009/TECH/04/13/twitter.worm/, April 2009. Last Accessed on November 10, 2010 @ 16:17 EST.

[50] Inc. Cogent Real-Time Systems. *Documentation Library Cascade DataHub for Linux and QNX Version 6.4*. Cogent Real-Time Systems, Inc., 162 Guelph Street, Suite 253 Georgetown, Ontario Canada, L7G 5X7, 6.4-1 edition, September 2007. Downloaded: December 30, 2010, 19:00.

[51] Inc. Cogent Real-Time Systems. *Documentation Library Cascade Historian Version 6.4*. Cogent Real-Time Systems, Inc., 162 Guelph Street, Suite 253 Georgetown, Ontario Canada, L7G 5X7, 6.4-1 edition, September 2007. Downloaded: December 30, 2010, 19:00.

[52] Michael Cohen. PyFlag - PyFlagWiki. http://www.pyflag.net/cgi-bin/moin.cgi, April 2010. Last Accessed on January 4, 2011, 14:02 EST.

[53] Creative Commons. Attack Database. http://xssdb.dabbledb.com/publish/attackdb/dc23ad51-25ef-4fdc-92be-4a7cb606387e/xssdb.html. Last Accessed on October 10, 2010.

[54] Creative Commons. MediaWiki Main Page. http://www.mediawiki.org/wiki/MediaWiki, January 2007. Last Accessed on November 10, 2010 @ 20:56 EST.

[55] Creative Commons. Wikipedia Main Page. www.wikipedia.org, May 2010. Last Accessed on November 10, 2010 @ 21:01 EST.

[56] Wikimedia Commons. Opera (Web Browser). http://en.wikipedia.org/wiki/Opera_%28web_browser%29. Last Accessed on October 11, 2010.

[57] OWASP Community. SQL Injection - OWASP. http://www.owasp.org/index.php/SQL_Injection, March 2010. Last Accessed on March 12, 2011 at 13:37 PM.

[58] OWASP Application Security Community. Error Handling, Auditing and Logging. http://www.owasp.org/index.php/Error_Handling,_Auditing_and_Logging#Forensics_evidence. Accessed on 18/01/2010.

[59] OWASP Application Security Community. Main Page OWASP.org Website. http://www.owasp.org/index.php/Main_Page. Accessed on 18/01/2010.

[60] Pentaho Community. Verifying JasperReports Integration in the Pentaho Platform. http://wiki.pentaho.com/display/ServerDoc1x/4.+Verifying+JasperReports+Integration+in+the+Pentaho+Platform, August 2007. Last Accessed on January 5, 2011 1:00 EST.

[61] Chinotec Technologies Company. Parosproxy.org - Web Application Security. http://www.parosproxy.org/, January 2004. Last Accessed: April 17, 2011 at 16:09 EST.

[62] United States Congress, editor. *Gramm-Leach-Bliley Act*, number 106-102 in 113 Stat. 1338, 732 North Capitol Street, NW, Washington, DC 20401-0001, November 1999. 106th Congress, Senate and House of Representatives of the United States of America in Congress, United States Government Printing Office. Last Accessed: April 13, 2011 at 18:36 EST.

[63] US Congress, editor. *Health Insurance Portability and Accountability Act of 1996*, number 3103 in 104-191, 732 North Capitol Street, NW, Washington, DC 20401-0001, August 1996. Senate and House of Representatives of the United States of America in Congress, United States Government Printing Office. Last Accessed: April 13, 2011 at 17:57 EST.

[64] NEWMAN Services Corp. LogJam - Web Traffic Analysis - About LogJam. http://newmanservices.com/logjam/pages/about.asp, January 2002. Last Accessed 1/5/2011 9:44:00 EST.

[65] NEWMAN Services Corp. LogJam - Web Traffic Analysis - Step 1 - The Jammer. http://newmanservices.com/logjam/pages/tour1.asp, January 2002. Last Accessed 1/5/2011 9:47:00 EST.

[66] Novell Corp. An Introduction to Novell's Internet Caching System. *Articles and Tips*, 0:1, July 1999. Last Accessed: April 12, 2011 at 8:51 EST.

[67] ZOHO Corp. IBM AS/400 Log Management. http://www.manageengine.com/products/eventlog/ibm-as-400-log-management.html, January 2011. Last Accessed: April 13, 2011 at 12:20 EST.

[68] Adobe Corporation. Adobe Main Page. http://www.adobe.com/, January 2011. Last Accessed: April 4, 2011 at 20:47 EST.

[69] Apple Corporation. Mac OS X Technology Overview. http://developer.apple.com/technologies/mac/, January 2011. Last Accessed: April 13, 2011 at 20:22 EST.

[70] Facebook Corporation. Hiphop-PHP. https://github.com/facebook/hiphop-php/wiki/, January 2011. Last Accessed: April 7, 2011 at 10:01 EST.

[71] Hewlett Packard Corporation. HP - United States. http://welcome.hp.com/country/us/en/cs/home.html, January 2011. Last Accessed: April 7, 2011 at 10:30 EST.

[72] IBM Corporation. IBM - Canada. http://www.ibm.com/ca/en/, January 2011. Last Accessed: April 13, 2011 at 11:59 EST.

[73] Microsoft Corporation. Rich Text Format (RTF) Specification, Version 1.6. http://msdn.microsoft.com/en-us/library/aa140277%28v=office.10%29.aspx, May 1999. Last Accessed: April 13, 2011 at 22:16 EST.

[74] Microsoft Corporation. Windows History - Internet Explorer History. http://www.microsoft.com/windows/WinHistoryIE.mspx, June 2003.

[75] Microsoft Corporation. How to Monitor the DHCP Log File. http://support.microsoft.com/kb/298367, October 2006. Last Accessed: April 13, 2011 at 10:54 EST.

[76] Microsoft Corporation. Microsoft .NET Framework. http://www.microsoft.com/net/, January 2009. Last Accessed: April 15, 2011 at 18:13 EST.

[77] Microsoft Corporation. About Dynamic Data Exchange. http://msdn.microsoft.com/en-us/library/ms648774.aspx, January 2011. Last Accessed: April 15, 2011 at 17:46 EST.

[78] Microsoft Corporation. ETW Tracing. http://msdn.microsoft.com/en-us/library/ms751538.aspx, January 2011. Last Accessed: April 14, 2011 at 15:31 EST.

[79] Microsoft Corporation. Internet Explorer - Microsoft Windows. http://windows.microsoft.com/en-US/internet-explorer/products/ie/home, January 2011. Last Accessed: April 1, 2011 at 19:55 EST.

[80] Microsoft Corporation. MDAC. http://msdn.microsoft.com/en-us/data/aa937729.aspx, January 2011. Last Accessed: April 13, 2011 at 23:00 EST.

[81] Microsoft Corporation. Microsoft Corporation: Software, Smartphones, Online, Games, Cloud Computing, IT Business Technology, Downloads. http://www.microsoft.com/en-us/default.aspx, January 2011. Last Accessed: April 4, 2011 at 21:08 EST.

[82] Microsoft Corporation. Microsoft Excel 2010 - Microsoft Office. http://office.microsoft.com/en-ca/excel/, January 2011. Last Accessed: April 13, 2011 at 22:04 EST.

[83] Microsoft Corporation. Microsoft Word 2010 - Microsoft Office. http://office.microsoft.com/en-ca/word/, January 2011. Last Accessed: April 13, 2011 at 22:04 EST.

[84] Microsoft Corporation. MS-DOS. http://technet.microsoft.com/en-us/library/cc743186.aspx, January 2011. Last Accessed: April 7, 2011 at 20:40 EST.

[85] Microsoft Corporation. Network Monitor. http://technet.microsoft.com/en-us/library/cc938655.aspx, January 2011. Last Accessed: April 14, 2011 at 15:28 EST.

[86] Microsoft Corporation. SQL Server Online Resources: CTP, Troubleshooting |Microsoft MSDN. http://msdn.microsoft.com/en-us/sqlserver/aa336270, January 2011. Last Accessed: April 13, 2011 at 10:39 EST.

[87] Microsoft Corporation. The Basic MDX Query (MDX). http://msdn.microsoft.com/en-us/library/ms144785.aspx, January 2011. Last Accessed: April 16, 2011 at 19:10 EST.

[88] Microsoft Corporation. The Official Microsoft IIS Website. http://www.iis.net, January 2011. Last Accessed: April 1, 2011 at 19:58 EST.

[89] Microsoft Corporation. VBScript User's Guide. http://msdn.microsoft.com/en-us/library/sx7b3k7y%28v=vs.85%29.aspx, January 2011. Last Accessed: April 18, 2011 at 16:29 EST.

[90] Microsoft Corporation. Windows 7 - Microsoft Windows. http://windows.microsoft.com/en-US/windows7/products/home, January 2011. Last Accessed: April 4, 2011 at 14:20 EST.

[91] Microsoft Corporation. Windows 95. http://support.microsoft.com/gp/w95, January 2011. Last Accessed: April 4, 2011 at 21:38 EST.

[92] Microsoft Corporation. Windows 95 Architecture Components. http://technet.microsoft.com/en-ca/library/cc751120.aspx, January 2011. Last Accessed: April 16, 2011 at 23:04 EST.

[93] Microsoft Corporation. Windows Server 2003 Technical Documentation, Downloads and Additional Resources. http://technet.microsoft.com/en-us/windowsserver/bb512919.aspx, January 2011. Last Accessed: April 7, 2011 at 10:40 EST.

[94] Microsoft Corporation. Windows XP Home Page. http://www.microsoft.com/windows/windows-xp/default.aspx, January 2011. Last Accesed: April 7, 2011 at 10:32 EST.

[95] Mozilla Corporation. Mozilla Foundation Security Advisory 2009-22. http://www.mozilla.org/security/announce/2009/mfsa2009-22.html, 04 2009.

[96] Mysql & Oracle Corporation. Main Page. http://mysql.he.net. Last Accessed October 11, 2010 /emphhttp://mysql.he.net/tech-resources/articles/introduction-to-mysql-55.html.

[97] Nintendo Corporation. Nintendo.ca. http://www.nintendo.ca/cgi-bin/usersite/display_info.cgi, January 2008. Last Accessed: April 4, 2011 at 14:55 EST.

[98] Nintendo Corporation. Nintendo.ca :: Nintendo DS. http://www.nintendo.ca/cgi-bin/usersite/display_info.cgi?pageNum=1&lang=en, January 2008. Last Accessed: April 4, 2011 at 15:02 EST.

[99] Nintendo Corporation. Nintendo.ca :: Nintendo Wii. http://www.nintendo.ca/cgi-bin/usersite/display_info.cgi?pageNum=13&lang=en, January 2008. Last Accessed: April 4, 2011 at 14:57 EST.

[100] Oracle Corporation. Listener Parameters (listener.ora). http://download.oracle.com/docs/cd/B10500_01/network.920/a96581/listener.htm#500386, January 2001-2002. Last Accessed: April 15, 2011 at 11:53 EST.

[101] Oracle Corporation. MySQL :: The World's Most Popular Open Source Database. http://www.mysql.com/, January 2010. Last Accessed: April 7, 2011 at 9:31 EST.

[102] ALBERTA COURTS. R. v. Cox, 2003 ABQB 212 (CanLII). http://www.canlii.org/en/ab/abqb/doc/2003/2003abqb212/2003abqb212.html, March 2003. Last Accessed March 19, 2011, at 20:56 PM EST.

[103] D. Crocker. Internet Mail Architecture. http://tools.ietf.org/html/rfc5598, July 2009. Last Accessed: April 11, 2011 at 19:05 EST.

[104] David H. Crocker. Standard for the Format of ARPA Internet Text Messages. http://www.ietf.org/rfc/rfc0822.txt, August 1982. Last Accessed: April 20, 2011 at 13:12 EST.

[105] Michael Cross, Steven Kapinos, Haroon Meer, Igor Muttik PhD, Steve Palmer, Petko pdp D. Petkov, Roger Shields, and Roelof Temmingh and. *Web Application Vulnerabilities*. Syngress, Syngress Publishing, Inc. Elsevier, Inc. 30 Corporate Drive Burlington, MA 01803, January 2007.

[106] Ward Cunningham. Wiki: What is Wiki? http://www.wiki.org/wiki.cgi?WhatIsWiki, June 2002. Last Accessed on November 10, 2010 @ 20:26 EST.

[107] Refsnes Data. W3 Schools Online Web Tutorials. http://www.w3schools.com, 1999 - 2010.

[108] Refsnes Data. Displaying XML with CSS. http://www.w3schools.com/Xml/xml_display.asp, 09 2010.

[109] Brian Totty David Gourley. *HTTP: The Definitive Guide*. O'Reilly Media, 2002. Chapter 1, Chapter 8.

[110] Huibert de Vries Henk de Vries Ilan Oshri. *Standards Battles in Open Source Software : The Case of Firefox*. PALGRAVE MACMILLAN, 2008. pg.23.

[111] DEC. Digital Equipment Corporation. http://research.microsoft.com/en-us/um/people/gbell/digital/DEC%201957%20to%20Present%201978.pdf, January 1972-1978. Last Accessed: April 20, 2011 at 12:35 EST.

[112] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPV6) Specification. http://tools.ietf.org/html/rfc2460, December 1998. Last Accessed: April 15, 2011 at 13:11 EST.

[113] Laurent Destailleur. AWStats - Free Log File Analyzer for Advanced Web Statistics (GNU GPL). - AWStats Logfile Analyzer 7.0 Documentation Frequently Asked Questions + Troubleshooting. http://awstats.sourceforge.net/docs/awstats_faq.html, 06 2010. References directed to: FAQ-SEC100, Last Accessed on December 28, 2010 10:19 EST.

[114] Laurent Destailleur. AWStats - Free Log File Analyzer for Advanced Web Statistics (GNU GPL). Main Page. http://www.awstats.org, 06 2010. Last Accessed on December 28,2010 10:18 EST.

[115] Laurent Destailleur. *AWStats Logfile Analyzer Documentation*. NLTechno, 7.0 edition, August 2010. Downloaded from http://awstats.sourceforge.net/docs/awstats.pdf December 28, 2010 11:20 EST, page 3.

[116] CORE Labs Research Development. Corelabs Site - What is CORE Wisdom? http://corelabs.coresecurity.com/index.php?module=Wiki&action=view&type=tool&name=Core_Wisdom, August 2002. Last Accessed on December 31, 2010 9:50 EST.

[117] die.net. strftime(3): Format Date / Time. http://linux.die.net/man/3/strftime. Last Accessed: April 6, 2011 at 20:31 EST.

[118] die.net. `chroot`(2): Change Root Directory - Linux Man Page. http://linux.die.net/man/2/chroot. Last Accessed: April 4, 2011 at 20:06 EST.

[119] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol. http://tools.ietf.org/html/rfc5246, August 2008. Last Accessed: April 7, 2011 at 8:00 EST.

[120] Dormando. Memcached - A Distributed Memory Object Caching System. http://memcached.org/, January 2009. Last Accessed: April 7, 2011 at 8:15 EST.

[121] W3C Working Draft. User Agent Accessibility Guidelines (UAAG) 2.0. http://www.w3.org/TR/UAAG20/#intro-def-ua, June 2010. Editors: James Allan, Texas School for the Blind and Visually Impaired Kelly Ford, Microsoft Jan Richards, Adaptive Technology Resource Centre, University of Toronto Jeanne Spellman, W3C/Web Accessibility Initiative.

[122] R. Droms. Dynamic Host Configuration Protocol. http://tools.ietf.org/html/rfc2131, March 1997. Last Accessed: April 13, 2011.

[123] Alf Eaton. Security Against SQL Injection in Wordpress. http://hublog.hubmed.org/archives/001654.html, 04 2008. Last Accessed on October 10, 2010.

[124] ECMA. Introducing JSON. http://www.json.org/, December 1999. Last Accessed: April 18, 2011 at 15:36 EST.

[125] Codd E.F., Codd S.B., and Salley C.T. Providing OLAP (On-line Analytical Processing) to User-Analysts: An IT Mandate. -, 0:31, September 1993. Last Accessed: April 12, 2011 at 7:50 EST.

[126] John Ellson, Emden Gansner, Yifan Hu, Arif Bilgin, and Dwight Perry. Graphviz - Graph Visualization Software. http://www.graphviz.org/, April 2011. Last Accessed: April 15, 2011.

[127] Ralf S. Engelschall. mod_ssl: The Apache Interface to OpenSSL. http://www.modssl.org/, January 1998-2001. Last Accessed: April 6, 2011 at 16:01 EST.

[128] Tobias Erbsland. ProfZone Anteater. http://anteater.sourceforge.net/, January 2004. Last Accessed on December 26, 2010 at 13:56 EST.

[129] erlang.org. Erlang Programming Language. http://www.erlang.org/, January 2011. Last Accessed: April 7, 2011 at 9:39 EST.

[130] T. N. Hastings et. al. American National Standard for Information Systems | Coded Character Sets | 7-Bit American National Standard Code for Information Interchange (7-Bit ASCII), March 1986. Individual metacharacters definitions found in a table on page 9.

[131] et al. Fielding. RFC 2616 - Hypertext Transfer Protocol – HTTP/1.1. http://tools.ietf.org/html/rfc2616#section-9, June 1999. Last Accessed: April 6, 2011 at 13:41 EST.

[132] Roy Fielding. wwwstat Manual. http://ftp.ics.uci.edu/pub/websoft/wwwstat/wwwstat.html, November 1996. Last Accessed on January 4, 2011 11:49 EST.

[133] Roy Thomas Fielding. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. Last Accessed: April 7, 2011 at 10:07 EST.

[134] Flowerfire. Sawmill Documentation - Technical Manual SSQL: Sawmill Structured Query Language (SQL). http://www.sawmill.net/cgi-bin/sawmill7/docs/sawmill.cgi?dp+docs.technical_

manual.ssql+webvars.username+samples+webvars.password+sawmill, January 2011. Last Accessed on: Fri 21 Jan 2011 09:38:59 AM EST.

[135] Flowerfire. Sawmill Documentation Technical Manual: Salang: the Sawmill Language. http://www.sawmill.net/cgi-bin/sawmill7/docs/sawmill.cgi?dp+docs.technical_manual.salang+webvars.username+samples+webvars.password+sawmill, January 2011. Last Accessed on: Fri 21 Jan 2011 10:18:19 AM EST.

[136] Lexum for The Federation of Law Societies of Canada. CanLII - Canadian Legal Information Institute. http://www.canlii.org/en/, April 2011. Last Accessed: April 17, 2011 at 12:55 EST.

[137] Internet Engineering Task Force. The Internet Engineering Task Force Main Page. http://www.ietf.org/, March 2011. Last Accessed: April 1, 2011 at 20:26 EST.

[138] Ben Forta. *Teach Yourself Regular Expressions in 10 Minutes*. Sams, Indianapolis, Indiana, 46240 USA, February 2004.

[139] Apache Software Foundation. Apache ObjectRelationalBridge - OJB. http://db.apache.org/ojb/, March 2006. Last Accessed on December 26, 2010, at 21:30 EST.

[140] Apache Software Foundation. Welcome to DdlUtils. http://db.apache.org/ddlutils/, June 2007. Last Accessed on December 26, 2010 at 21:35 pm.

[141] Apache Software Foundation. Apache Thrift. http://incubator.apache.org/thrift/, January 2007-2010. Last Accessed: April 7, 2011 at 9:49 EST.

[142] Apache Software Foundation. DB Apache Project - Welcome to DB. http://db.apache.org/, November 2010. Last Accessed on December 26, 2010 at 21:37 EST.

[143] Apache Software Foundation. Java Data Objects (JDO). http://db.apache.org/jdo/, July 2010. Last Accessed on December 26, 2010 at 21:23 EST.

[144] Apache Software Foundation. Torque - Torque. http://db.apache.org/torque/, March 2010. Last Accessed on December 26, 2010 at 21:19 EST.

[145] Apache Software Foundation. Apache Module mod_log_forensic. http://httpd.apache.org/docs/current/mod/mod_log_forensic.html, January 2011. Last Accessed: April 4, 2011 at 20:17 EST.

[146] Apache Software Foundation. Apache Module mod_proxy. http://httpd.apache.org/docs/2.0/mod/mod_proxy.html, January 2011. Last Accessed: April 4, 2011 at 20:13 EST.

[147] Apache Software Foundation. Apache Traffic Server. http://trafficserver.apache.org/, March 2011. Last Accessed: April 12, 2011 at 8:44 EST.

[148] Mozilla Foundation. Mozilla Features. http://www-archive.mozilla.org/why/users-features.html#standards, 07 2008.

[149] OPC Foundation. About OPC - What is OPC? http://www.opcfoundation.org/Default.aspx/01_about/01_whatis.asp?MID=AboutOPC, January 2010. Last Accessed on December 30, 2010 at 21:11 EST.

[150] OWASP Foundation. OWASP Webscarab Project. http://www.owasp.org/index.php/Category:OWASP_WebScarab_Project, January 2011. Last Accessed: April 4, 2011 at 14:00 EST.

[151] Python Software Foundation. Python Programming Language. http://www.python.org/, January 1990-2010. Last Accessed: April 6, 2011 at 10:58 EST.

[152] Stichting LogReport Foundation. Report Format Help From Lire Output. entered command `lr_xml2report --help report-templates`, June 2006. Lire Installed from Debian repositories, January 4, 20:08 EST.

[153] Stichting LogReport Foundation. Lire. http://www.logreport.org/lire.html, June 2007. Last Accessed on January 4, 2011 23:24 EST.

[154] The Apache Software Foundation. The Apache Cassandra Project. http://cassandra.apache.org/, January 2009. Last Accessed: April 7, 2011 at 9:54 EST.

[155] The Apache Software Foundation. Core - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/core.html#servername, January 2011. Last Accessed: April 6, 2011 at 21:18 EST.

[156] The Apache Software Foundation. Core - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/core.html#usecanonicalname, January 2011. Last Accessed: April 6, 2011 at 21:30 EST.

[157] The Apache Software Foundation. mod_isapi - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/mod_isapi.html, January 2011. Last Accessed: March 28, 2011 at 20:14 EST.

[158] The Apache Software Foundation. mod_log_config - Apache HTTP Server. http://httpd.apache.org/docs/current/mod/mod_log_config.html, January 2011. Last Accessed: April 6, 2011 at 11:01 EST.

[159] The Apache Software Foundation. mod_log_config - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/mod_log_config.html#logformat, January 2011. Last Accessed: April 6, 2011 at 14:12 EST.

[160] The Apache Software Foundation. mod_logio - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/mod_logio.html, January 2011. Last Accessed: April 6, 2011 at 14:16 EST.

[161] The Apache Software Foundation. mod_session - Apache HTTP Server. http://httpd.apache.org/docs/2.3/mod/mod_session.html, January 2011. Last Accessed: April 6, 2011 at 18:03 EST.

[162] The Apache Software Foundation. mod_usertrack - Apache HTTP Server. http://httpd.apache.org/docs/2.0/mod/mod_usertrack.html, January 2011. Last Accessed: April 6, 2011 at 18:01 EST.

[163] The Apache Software Foundation. Welcome! - The Apache HTTP Server Project. http://httpd.apache.org/, January 2011. Last Accessed: March 28, 2011, 19:24 EST.

[164] The Apache Software Foundation. Welcome to the Apache Software Foundation! http://www.apache.org/, January 2011. Last Accessed: March 28, 2011 at 19:12 EST.

[165] The Eclipse Foundation. Eclipse - The Eclipse Foundation Open Source Community Website. http://www.eclipse.org/, January 2011. Last Accessed: April 16, 2011 at 18:29 EST.

[166] Brian Fox and Chet Ramey. *`history` - GNU History Library.* Free Software Foundation and Case Western Reserve University, Boston, MA, 3 edition, July 2003.

[167] Jeffrey E. F. Friedl. *Mastering Regular Expressions.* O'Reilly Media, Sebastopol, CA 95472, 3rd edition, August 2006.

[168] FSF. *`ac` - Print Statistics About Users' Connect Time.* Free Software Foundation, Boston, MA, 2 edition, October 1995.

[169] FSF. *`lastcomm` - Print Out Information About Previously Executed Commands.* Free Software Foundation, Boston, MA, 2 edition, October 1995.

[170] FSF. *Xferlog - HylaFAX Activity Log*, June 1996. Last Accessed: April 15, 2011 at 9:30 EST.

[171] FSF. *`sa` - Summarizes Accounting Information.* Free Software Foundation, Boston, MA, 2 edition, August 1997.

[172] FSF. *grep(1) User Commands.* Free Software Foundation Inc., Boston, MA USA, gnu grep 2.5.1-cvs edition, February 2008.

[173] FSF. *`acct` - Switch Processing Accounting On or Off.* Free Software Foundation, Boston, MA, 2 edition, July 2008.

[174] FSF. *`accton` - Turns Procss Accounting On or Off.* Free Software Foundation, Boston, MA, 2 edition, November 2008.

[175] FSF. *Utmp, Wtmp - Login Records.* Linux Man Pages Project, FSF, Boston, MA USA, 5 edition, October 2008.

[176] Bernardo Damele A. G. Command Execution with a MySQL UDF. http://bernardodamele.blogspot.com/2009/01/command-execution-with-mysql-udf.html, January 2009. Last Accessed: March 21, 2011 at 17:59 PM EST.

[177] Simson Garfinkel and Gene Spafford. *Practical UNIX and Internet security.* Computer security Computer security (Sebastopol, Calif.). O'Reilly & Associates, Inc., pub-ORA:adr, second (completely rewritten and expanded to include Internet security) edition, 1996.

[178] Inc. Geeknet. Logrep | Freshmeat. http://freshmeat.net/projects/logrep/, January 2011. Last Accessed on January 4, 2011 14:27 EST.

[179] R. Gerhards. The Syslog Protocol. http://tools.ietf.org/html/rfc5424, March 2009. Last Accessed: April 13, 2011 at 12:27 EST.

[180] Vivek Gite. How to Keep a Detailed Audit Trail of What's Being Done on Your Linux Systems. http://www.cyberciti.biz/tips/howto-log-user-activity-using-process-accounting.html, November 2006. Last Accessed on 12-24-2010 at 18:26 EST.

[181] Vivek Gite. Understanding /var/account/pacct OR /var/account/acct Acct File Format. http://www.cyberciti.biz/faq/linux-unix-bsd-varaccountpacct-or-varlogaccountpacct-file/, March 2008. Last Accessed on 2010.12.24 at 10:17 EST.

[182] Gabriele Giuseppini. *Microsoft Log Parser Toolkit*. Syngress, Syngress Publishing, Inc. 800 Hingham Street Rockland, MA 02370, 2004.

[183] Gabriele Giuseppini. *Microsoft Log Parser Toolkit*. Syngress Publishing, first edition, 2005.

[184] Intevation GmbH. OpenVAS - Open Vulnerability Assessment System Community Site. http://www.openvas.org/, January 2011. Last Accessed: April 10, 2011 at 22:34 EST.

[185] goofyyasd.dhs.org. Apachedb. http://yasd.cc/en/apachedb.php3, June 2003. Last Accessed at 8:18, June 6 2003.

[186] James Gosling, Bill Joy, Guy Steele, and Gilad Bracha. *The Java Language Specification*. Addison - Wesley, Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054 U.S.A., 3rd edition, May 1996 - 2005. Last Accessed:.

[187] Jan Goyvaerts. Regular Expressions Reference - Advanced Syntax. http://www.regular-expressions.info/refadv.html, October 2010.

[188] Jan Goyvaerts. Regular Expressions Reference - Basic Syntax. http://www.regular-expressions.info/reference.html, October 2010.

[189] Emilio Grimaldo. Managing Your Logs With `chklogs`. http://www.linuxjournal.com/node/2363/print, April 1998. Last Accessed on December 31, 2010, 8:04 EST.

[190] Jeremiah Grossman, Robert "RSnake" Hansen, Petko "pdp" D. Petkov, and Anton Rager. *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress, Syngress Publishing, Inc. Elsevier, Inc. 30 Corporate Drive Burlington, MA 01803, January 2007.

[191] Miniwatts Marketing Group. Internet World Stats Usage and Population Statistics. http://www.internetworldstats.com/stats.htm. Accessed on 18/01/2010.

[192] PHPIDS Group. Main Page. http://php-ids.org/. Last Accessed on October 11, 2010.

[193] The Austin Group. The Open Group Base Specifications Issue 7, IEEE Std 1003.1-2008. http://www.unix.org/single_unix_specification/, January 2008. Last Accessed March 28, 2011 at 18:46 EST.

[194] The PHP Group. PHP: Hypertext Preprocessor. http://www.php.net/, January 2001-2011. Last Accessed: April 7, 2011 at 9:27 EST.

[195] The PHP Group. PHP: Possible Modifiers in Regex Patterns - Manual. http://www.php.net/manual/en/reference.pcre.pattern.modifiers.php, November 2010. Last Accessed on November 15, 2010.

[196] The Shmoo Group. Osiris User Handbook. http://osiris.shmoo.com/handbook.html, January 2005. Last Accessed on February 22, 2011.

[197] Inc. Guidance Software. Encase |Cyber Security |Computer Forensics |Network Security |E-Discovery |eDiscovery |Forensics |Forensic. http://www.guidancesoftware.com/, January 2011. Last Accessed: April 17, 2011 at 12:46 EST.

[198] Phillip M. Hallam-Baker and Brian Behlendorf. Extended Log File Format. http://www.w3.org/TR/WD-logfile.html, February 1999. Last Accessed: April 6, 2011 at 14:29 EST.

[199] M. Handley and E. Rescorla. Internet Denial-of-Service Considerations. http://tools.ietf.org/html/rfc4732, November 2006. Last Accessed: April 4, 2011 at 19:18 EST.

[200] L.P. Hewlett-Packard Development Company. Compaq TaskSmart Internet Caching Server Series. http://h20000.www2.hp.com/bizsupport/TechSupport/Home.jsp?lang=en&cc=us&prodTypeId=15351&prodSeriesId=254622&submit.y=5&submit.x=4&lang=en&cc=us, January 2011. Last Accessed: April 12, 2011 at 8:39 EST.

[201] L.P. Hewlett-Packard Development Company. HP-UX 11iHP UNIX ®Powers the Mission-critical Converged Infrastructure. http://h71028.www7.hp.com/enterprise/w1/en/os/hpux11i-overview.html, January 2011. Last Accessed: April 16, 2011 at 23:52 EST.

[202] Hobbit. mod_chroot. http://core.segfault.pl/~hobbit/mod_chroot/index.html, June 2008. Last Accessed: April 4, 2011 at 16:21 EST.

[203] B. Hoehrmann. RFC 4329 - Scripting Media Types. http://tools.ietf.org/html/rfc4329, April 2006. Last Accessed: April 3, 2011 at 20:52 EST.

[204] horde.org. Horde - The Horde Project. http://www.horde.org/apps/horde/, January 2011. Last Accessed: April 7, 2011 at 18:20 EST.

[205] httpd@w3.org. Logging in w3c httpd. http://www.w3.org/Daemon/User/Config/Logging.html, July 1995.

[206] IBM. IBM Power Systems Software - AIX: Overview. http://www-03.ibm.com/systems/power/software/aix/index.html, January 2011. Last Accessed: April 16, 2011 at 23:56 EST.

[207] IEEE and The Open Group. Fork - Create a New Process. http://pubs.opengroup.org/onlinepubs/9699919799/functions/fork.html, January 2008. Last Accessed: April 10, 2011 at 19:53 EST.

[208] ArcSight Inc. Administrator's Guide ArcSight Logger^{TM}v5.0, Patch2, November 2010. Page 4, Downloaded December 26, 2010.

[209] ArcSight Inc. Product Brief: ArcSight ESM Single Platform for Enterprise-Wide Visibility. http://www.arcsight.com/collateral/briefs/ArcSight_ProductBrief_ESM.pdf, 2010. Page 2.

[210] ArcSight Inc. Product Brief: ArcSight Logger Universal Log Management Solution. http://www.arcsight.com/collateral/briefs/ArcSight_ProductBrief_Logger.pdf, 2010. Page 2.

[211] Cisco Systems Inc. Application Networking Services - Main Page - Cisco Systems. http://www.cisco.com/en/US/products/hw/contnetw/index.html, January 2011. Last Accessed: April 12, 2011 at 8:35 EST.

[212] Cisco Systems Inc. Cisco Systems Homepage. http://www.cisco.com/, April 2011. Last Accessed: April 12, 2011.

[213] Cogent Real-Time Systems Inc. Cogent Real-Time Systems - Data Connectivity Software. http://www.cogent.ca/Software/DataHub.html, January 2010. Last Accessed on December 30, 2010, 18:16 EST, Page 1.

[214] Flowerfire Inc. Apache Error Log Analyzer. http://www.sawmill.net/formats/apache_error.html, 2010.

[215] Linux Online Inc. Linux Online - Application: apachedb. http://www.linux.org/apps/AppId_1324.html, January 2008. Last Accessed on December 26, at 19:33 EST.

[216] Quest Software Inc. Funnel Web Analyzer 5.0 Release Notes. http://www.quest.com/funnel_web/payload/FWAReleaseNotes.htm, March 2003. Last Accessed on January 4, 2011, 18:50 EST.

[217] Red Hat Inc. Cygwin. http://www.cygwin.com/, January 2000-2011. Last Accessed: April 16, 2011 at 23:01 EST.

[218] Talend Inc. Talend - Talend Open Studio. http://www.talend.com/products-data-integration/talend-open-studio.php, January 2010. Last Accessed on December 28, 2010, 13:11 EST.

[219] Adobe Systems Incorporated. Adobe - Adobe Reader Download. http://get.adobe.com/reader/, January 2011. Last Accessed: April 5, 2011 at 13:35 EST.

[220] Adobe Systems Incorporated. Flash Developer Center. http://www.adobe.com/devnet/flash.html, January 2011. Last Accessed: April 5, 2011 at 13:33 EST.

[221] America Online Incorporated. AOL.ca - Canada's Breaking News, Entertainment, Music Life & Style and Email. http://www.aol.com, January 2011. Last Accessed: April 6, 2011 at 10:08 EST.

[222] CNN International. 'Bridget Jones' Blogger Fire Fury. http://edition.cnn.com/2006/WORLD/europe/07/19/france.blog/index.html?section=cnn_tech, July 2006. Last Accessed on November 9, 2010, @ 23:13 EST.

[223] SUPREME COURT OF PRINCE EDWARD ISLAND. R. v. Harris, 2010 PESC 32 (CanLII). http://www.canlii.org/en/pe/pesctd/doc/2010/2010pesc32/2010pesc32.html, June 2010. Last Accessed on March 19, 2011, at 20:59 PM EST.

[224] itripn, Paul Herman stephd, and David LaPalomento. Open Source Tripwire. http://sourceforge.net/projects/tripwire/, April 2011. Last Accessed: April 15, 2011 at 12:33 EST.

[225] G. Jakobson and M. Weissman. Alarm correlation. *Network, IEEE*, 7(6):52–59, 1993.

[226] JasperForge. JasperForge ¿ iReport Project Wiki. http://jasperforge.org/plugins/mwiki/index.php/Ireport/What_is_iReport, 2000-2010. Last Accessed on January 5, 2011 00:58 EST.

[227] Amos Jeffries. Squid: Optimising Web Delivery. http://www.squid-cache.org/, September 2010. Last Accessed on: Thu 27 Jan 2011 02:10:58 AM EST.

[228] M. St. Johns. Identification Protocol. http://www.apps.ietf.org/rfc/rfc1413.html, February 1993. Last Accessed: April 6, 2011 at 11:13 EST.

[229] Ron Jones, editor. *Sarbanes-Oxley Act*, number 3763, PO Box 210040 Clifton Avenue & Calhoun Street Cincinnati, OH 45221-0040, January 2002. One Hundred Seventh Congress of the United States of America, University of Cincinnati College of Law. Last Accessed: April 13, 2011 at 16:50 EST.

[230] Simon Josefsson. The Base16, Base32, and Base64 Data Encodings. http://tools.ietf.org/html/rfc4648, October 2006. Last Accessed: April 18, 2011 at 16:12 EST.

[231] Chris Josephes. Writing Apache's Logs to Mysql - O'Reilly Media. http://onlamp.com/pub/a/apache/2005/02/10/database_logs.html, February 2005. Last Accessed on December 26, 2010, at 21:11 EST.

[232] Audun Jsang, Bander AlFayyadh, Tyrone Grandison, Mohammed AlZomai, and Judith McNamara. Security Usability Principles for Vulnerability Analysis and Risk Assessment. *Computer Security Applications Conference, Annual*, 0:269–278, 2007.

[233] Ed. K. Zeilenga. Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map. http://tools.ietf.org/html/rfc4510, June 2006. Last Accessed: April 18, 2011 at 12:49 EST.

[234] Poul-Henning Kamp, Robert Watson, Bjoern A. Zeeb, Pawel Jakub Dawidek, and James Gritton. FreeBSD Man Pages - Jail. http://www.freebsd.org/cgi/man.cgi?query=jail, January 2010. Last Accessed: April 4, 2011 at 20:10 EST.

[235] Tevfik Karagulle. Logrep. http://sourceforge.net/projects/logrep/, March 2006. Last Accessed on January 4, 2011 at 14:25 EST.

[236] Amit Klein. HTTP Response Smuggling. whitepaper, SecurityFocus, aksecurityhotpop.com, February 2006.

[237] Amit Klein, Chaim Linhart, Ronen Heled, and Steve Orrin. HTTP Request Smuggling. whitepaper, Watchfire Corp., aksecurityhotpop.com, February 2005.

[238] John C. Klensin. Simple Mail Transfer Protocol. http://tools.ietf.org/html/rfc5321, October 2008. Last Accessed: April 17, 2011 at 00:18 EST.

[239] G. Klyne and C. Newman. Date and Time on the Internet: Timestamps. http://tools.ietf.org/html/rfc3339, July 2002. Last Accessed: April 10, 2011 at 16:59 EST.

[240] Damon Kohler. Email Injection. http://www.damonkohler.com/2008/12/email-injection.html, 09 2010.

[241] Jukka Korpela. Tab Separated Values (TSV): a Format for Tabular Data Exchange. http://www.cs.tut.fi/~jkorpela/TSV.html, February 2005. Last Accessed: April 14, 2011.

[242] Dinko Korunic. Index of /projects/gwstat. http://dkorunic.net/projects/gwstat/, December 2004. Last Accessed: April 13, 2011 at 20:05 EST.

[243] Mrs K.Velli and Mr E.Melagrakis. Information Technology – 8-bit Single-Byte Coded Graphic Character Sets – Part 1: Latin Alphabet No. 1. http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=28245, February 1998. Last Accessed: April 7, 2011 at 20:58 EST.

[244] Samuli Krkkinen. *README from weblogmon-0.1.2*. woods.iki.fi, skarkkai@woods.iki.fi, 0.1.2 edition, June 2000. Last accessed January 5, 2011 20:55 EST.

[245] Samuli Krkkinen. Weblogmon Homepage. http://weblogmon.sourceforge.net/, November 2001. Last Accessed on January 5, 2011 20:12 EST.

[246] NLnet Labs. Net::DNS and Net::DNS::SEC. http://www.net-dns.org/, April 2011. Last Accessed: April 17, 2011 at 10:23 EST.

[247] Felipe Micaroni Lalli. `Web_development_timeline.png`. http://upload.wikimedia.org/wikipedia/commons/e/e4/Web_development_timeline.png, 2008. Accessed on 18/01/2010.

[248] Thor Larholm. Internet Explorer 0-day Exploit. http://larholm.com/2007/07/10/, 06 2007. last accessed on October 11, 2010.

[249] Ben Laurie and Peter Laurie. *Apache: The Definitive Guide*. O'Reilly & Associates, Sebastopol, California, March 1997.

[250] Chris Lavigne. Breadboard BI Web Analytics. http://sourceforge.net/projects/web-analytics/, October 2007. Last Accessed on December 28, 2010 at 13:04 EST.

[251] Rami Lehti, Pablo Virolainen, Richard van den Berg, and Hannes von Haugwitz. AIDE - Advanced Intrusion Detection Environment. http://aide.sourceforge.net/, March 2011. Last Accessed: April 15, 2011 at 12:31 EST.

[252] Jan Goyvaerts Steven Levithan. *Regular Expressions Cookbook*. O'Reilly Media, Sebastopol, CA 95472, 1st edition, May 2009.

[253] a division of Reed Elsevier Inc. LexisNexis. LexisNexis Canada. http://www.lexisnexis.ca/en/, January 2011. Last Accessed: April 17, 2011 at 12:53 EST.

[254] John Leyden. Twitter Blames Website Upgrade for Re-introducing XSS Hole. http://www.theregister.co.uk/2010/09/22/twitter_xss_genesis/, Septemeber 2010. Last Accessed on November 10, 2010 @ 16:05 EST.

[255] Backtrack Linux. Backtrack Linux - Penetration Testing Distribution. http://www.backtrack-linux.org/, January 2011. Last Accessed: April 10, 2011 at 22:31 EST.

[256] ArcSight LLC. Common Event Format - ArcSight. http://www.arcsight.com/solutions/solutions-cef/, January 2011. Last Accessed: April 16, 2011 at 20:37 EST.

[257] Elsevier Ltd. Tomorrow's Microsoft Patch Tuesday Biggest on Record. http://www.infosecurity-magazine.com/view/13119/tomorrows-microsoft-patch-tuesday-biggest-on-record/, 10 2010. Last Accessed on October 11, 2010.

[258] Michael W. Lucas. *PGP & GPG Email for the Practical Paranoid*. No Starch Press, No Starch Press, Inc. 555 De Haro Street, Suite 250, San Francisco, CA 94107, February 2006.

[259] Teresa F. Lunt. A Survey of Intrusion Detection Techniques. *Computers & Security*, 12:405–418, 1993.

[260] Ed. M. Smith and T. Howes. Lightweight Directory Access Protocol (LDAP): Uniform Resource Locator. http://tools.ietf.org/html/rfc4516#page-5, June 2006. Last Accessed on: March 12, 2011 at 8:06 AM.

[261] Seven Simple Machines. Report Magic. http://www.reportmagic.org/, January 2005. Last Accessed: April 11, 2011 at 14:09 EST.

[262] ManageEngine. *EventLog Analyzer 6 Managed Server Guide*. ZOHO Corporation, 4900 Hopyard Rd, Suite 310 Pleasanton, CA 94588, USA, build 6.2.0 edition, January 2009. Last Accessed on January 2, 2011 at 13:06 EST.

[263] Chris Green Martin Roesch. *Snort Users Manual 2.9.0 The Snort Project*. Sourcefire Inc., 9770 Patuxent Woods Drive Columbia, MD 21046 United States, 2.9.0 edition, December 2010. Last accessed on February 10, 2011.

[264] William McCutchen. Python strftime Reference. http://strftime.org/, August 2010. Last Accessed: April 6, 2011 at 20:34 EST.

[265] M. Mealling and R. Denenberg. Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations. http://tools.ietf.org/html/rfc3305, August 2002. Last Accessed: April 4, 2011 at 15:16 EST.

[266] J.D. Meier, Srinath Vasireddy, Michael Dunner, Ray Escamilla, and Anandha Murukan. *Microsoft Improving Web Application Security Threats and Countermeasures*. Microsoft Press, Corporate Headquarters Microsoft Corporation One Microsoft Way Redmond, WA 98052-6399, June 2003.

[267] Roger Meyer and Carlos Cid (advisor). Detecting Attacks on Web Applications from Log Files. In *Information Security Reading Room*, January 2008. ©SANS Institute 2008, As part of the Information Security Reading Room.

[268] Microsoft. How to View and Manage Event Logs in Event Viewer in Windows XP. http://support.microsoft.com/kb/308427, May 2007.

[269] Sun Microsystems. Enterprise JavaBeans Query Language. http://java.sun.com/j2ee/tutorial/1_3-fcs/doc/EJBQL.html, December 2007. Last Accessed: April 16, 2011 at 18:44 EST.

[270] Jelena Mirkovic, Sven Dietrich, David Dittrich, and Peter Reiher. *Internet Denial of Service: Attack and Defense Mechanisms*. Prentice Hall, December 2004. Chapter 2.

[271] P. Mockapetris. Domain Names - Implementation and Specification. http://tools.ietf.org/html/rfc1035, November 1987. Last Accessed: April 14, 2011 at 15:07.

[272] Mozilla. Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/. Last Accessed on: April 4, 2011 at 14:35 EST.

[273] Mozilla. Mozilla | Firefox Web Browser & Thunderbird Email Client. http://www.mozilla.com/en-US/firefox/new/. Last Accessed: April 4, 2011 at 21:17 EST.

[274] Mozilla. Plugins :: Add-ons for Firefox. https://addons.mozilla.org/en-US/firefox/browse/type:7. Last Accessed on April 4, 2011 at 14:37 EST.

[275] Individual mozilla.org Contributors. Home of the Mozilla Project. http://www.mozilla.org/, January 1998-2011. Last Accessed: April 4, 2011 at 21:14 EST.

[276] Mozillazine.org. Security Exploit Uses Internet Explorer to attack Mozilla Firefox. http://www.mozillazine.org/talkback.html?article=22198, 07 2007. Accessed through google cache on October 11, 2010.

[277] David Murphy. Twitter: On-Track for 200 Million Users by Year's End. http://www.pcmag.com/article2/0,2817,2371826,00.asp, October 2010. Last Accessed on November 10, 2010 @ 15:29 EST.

[278] mustlive administrator at http://websecurity.com.ua. Cross-Site Scripting Vulnerability in Mozilla Firefox, Opera and Other Browsers. http://seclists.org/fulldisclosure/2010/Aug/85.

[279] Evi Nemeth, Garth Snyder, Trent R. Hein, and Ben Whaley. *UNIX and Linux System Administration Handbook*. Prentice Hall, 4th edition, July 2010.

[280] Netscape. Netscape and Sun Announce Javascript, the Open, Cross-Platform Object Scripting Language for Enterprise Networks and the Internet. http://web.archive.org/web/20070916144913/http://wp.netscape.com/newsref/pr/newsrelease67.html, December 1995. Last Accessed: April 3, 2011 at 20:26 EST.

[281] Oracle Sun Developer Network. Java 2 Platform, Enterprise Edition (J2EE) Overview. http://java.sun.com/j2ee/overview.html, January 2010. Last Accessed: April 15, 2011 at 18:15 EST.

[282] Mark Nottingham. Squidj. http://www.mnot.net/squij/, January 1998. Last Accessed on: January 27 at 18:18 PM EST.

[283] Directorate General of Budget. CNS 11643 Chinese Standard Interchange Code Master Ideographs Server. http://www.cns11643.gov.tw, January 2008. Last Accessed: April 20, 2011 at 11:59.

[284] Government of Canada. The Federal Information Technology Market. http://www.canadainternational.gc.ca/sell2usgov-vendreaugouvusa/opportunities-opportunites/it-info-ti.aspx?lang=eng, December 2010. Last Accessed: April 13, 2011 at 19:50 EST.

[285] Mustlive Administrator of http://websecurity.com.ua. Cross-Site Scripting Vulnerability in Mozilla, Firefox and Chrome. http://seclists.org/bugtraq/2009/Jul/94, 07 2009.

[286] Minister of Industry Canada. BILL C-28. http://www2.parl.gc.ca/HousePublications/Publication.aspx?Docid=4547728&file=4, May 2010. Last Accessed: March 15, 2011, at 15:52 PM.

[287] Board of Trustees of the University of Illinois. NCSA Software and Technologies. http://illinois.edu/lb/imageList/2943, January 2011. Last Accessed: April 6, 2011 at 14:40 EST.

[288] Joey Olson. *Linux Help - DHCP Setup Guide*. Private World Domination Inc., 306A-219 Dufferin St. Toronto, ON M6K 3J1 CA, December 2003. Last Accessed: April 13, 2011 at 11:38 EST.

[289] Cisco Security Intelligence Operations. What is the Difference: Viruses, Worms, Trojans, and Bots? http://www.cisco.com/web/about/security/intelligence/virus-worm-diffs.html, January 2011. Last Accessed: April 10, 2011 at 10:12 EST.

[290] Oracle. Oracle Berkeley DB. http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html, January 2011. Last Accessed: April 16, 2011 at 22:33 EST.

[291] Oracle. Oracle Database Online Documentation 10g Release 2 (10.2). http://www.oracle.com/pls/db102/homepage, January 2011. Last Accessed: April 13, 2011 at 12:30 EST.

[292] Oracle. Oracle PL/SQL. http://www.oracle.com/technetwork/database/features/plsql/index.html, January 2011. Last Accessed: April 16, 2011 at 19:25 EST.

[293] Oracle. Oracle Solaris. http://www.oracle.com/us/products/servers-storage/solaris/index.html, April 2011. Last Accessed: April 13, 2011 at 20:20 EST.

[294] Oracle and Project Kendai. OpenOffice.org - The Free and Open Productivity Suite. http://www.openoffice.org/, April 2011. Last Accessed: April 16, 2011 at 19:43 EST.

[295] Oracle and Sun. Web Tier. http://www.oracle.com/us/products/middleware/application-server/050735.html, January 2011. Last Accessed: April 12, 2011 at 8:48 EST.

[296] Tim O'Reilly. What is Web 2.0. http://oreilly.com/web2/archive/what-is-web-20.html, September 2005. Last Accessed on November 20, @ 22:36 EST.

[297] Tavis Ormandy. Microsoft Windows Help Centre Handles Malformed Escape Sequences Incorrectly. http://seclists.org/fulldisclosure/2010/Jun/205, June 2010. Last Accessed on November 10, 2010 @ 18:47 EST.

[298] OSSEC. File Monitoring - OSSEC v2.5.0 Documentation. http://www.ossec.net/doc/manual/monitoring/file-log-monitoring.html, January 2010. Last Accessed on January 6, 2011 15:07 EST.

[299] OSVDB. 64146 : HP System Management Homepage (SMH) red2301.html Redirect Url Parameter Arbitrary Site Redirect. http://osvdb.org/show/osvdb/64146, April 2010. Last Accessed on November 10, 2010 @ 18:51 EST.

[300] OWASP. Argument Injection or Modification. http://www.owasp.org/index.php/Argument_Injection_or_Modification, 5 2009. Last Accessed on: Thursday March 10, 2011 21:56 pm.

[301] OWASP. OWASP Top 10 - 2010 The Ten Most Critical Web Application Security Risks. Technical report, OWASP, OWASP Foundation 9175 Guilford Rd Suite 300 Columbia, Maryland 21046 United States, April 2010. Last Accessed: March 13, 2010 at 16:23 PM.

[302] Paradox. An Introduction to CSRF Attacks. *2600 Magazine*, 27(1):30–31, Spring 2010.

[303] Canadian Parliament, editor. *Personal Information Protection and Electronic Documents Act*, number 362, 112 Kent Street Place de Ville Tower B, 3rd Floor Ottawa, Ontario K1A 1H3, April 2000. Thirty-sixth Parliament, 48-49 Elizabeth II, Office of the Privacy Commissioner of Canada. Last Accessed: April 13, 2011 at 18:22 EST.

[304] Charalampos Patrikakis, Michalis Masikos, and Olga Zouraraki. Distributed Denial of Service Attacks. *The Internet Protocol Journal*, 7(4):0, December 2004. Last Accessed: April 4, 2011 at 19:53 EST.

[305] LLC. PCI Security Standards Council. Official PCI Security Standards Council Site - Verify PCI Compliance, Download Data Security and Credit Card Security Standards. https://www.pcisecuritystandards.org/, January 2006-2011. Last Accessed: April 17 at 00:23 EST.

[306] MA USA 01880 PCI Security Standards Council, LLC 401 Edgewater Place Suite 600 Wakefield. Payment Card Industry Data Security Standard Requirements and Security Assessment Procedures. https://www.pcisecuritystandards.org/documents/pci_dss_v2.pdf, October 2010. Last Accessed: April 13, 2011 at 17:06 EST.

[307] Steven Pemberton, Daniel Austin, Jonny Axelsson, Tantek elik, Doug Dominiak, Herman Elenbaas, Beth Epperson, Masayasu Ishikawa, Shin'ichi Matsui, Shane McCarron, Ann Navarro, Subramanian Peruvemba, Rob Relyea, Sebastian Schnitzenbaumer, and Peter Stark. W3C XHTML 1.0 The Extensible HyperText Markup Language. http://www.w3.org/TR/xhtml1/, August 2002. Last Accessed March 13, 2011 21:15 PM.

[308] Perl.org. The Perl Programming Language. http://www.perl.org/, January 2002-2011. Last Accessed: April 6, 2011 at 10:56 EST.

[309] Inc.. Peter Thoeny, Twiki. TWiki - the Open Source Enterprise Wiki and Web 2.0 Application Platform. http://www.twiki.org/, March 2011. Last Accessed on: March 28, 2011 at 20:36 EST.

[310] Andrew Phelan. Couple Left Sickened Over Child Porn Sent to Xbox. http://www.herald.ie/national-news/couple-left-sickened-over-child-porn-sent-to-xbox-2105987.html, March 2010. Last Accessed: March 15, 2011.

[311] Michael Pophal Philipp Frauenfelder, Cord Beerman. *Calamaris Manpage*. Free Software Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA., 1 edition, March 2006. Last Accessed from a Debain copy of calamaris v. 2.99.4.0 installed December 28, 2010, 14:46 EST.

[312] William Enck Patrick Traynor Patrick McDaniel Thomas La Porta. Exploiting Open Functionality in SMS-Capable Cellular Networks. *Journal of Computer Security*, 16:713 – 742, 2008. http://www.smsanalysis.org/smsanalysis.pdf Last Accessed on: August 4, 2007.

[313] J. Postel. RFC 768 User Datagram Protocol. http://tools.ietf.org/html/rfc768, August 1980. Last Accessed: April 6, 2011 at 19:12 EST.

[314] J. Postel and J. Reynolds. File Transfer Protocol (FTP). http://tools.ietf.org/html/rfc959, October 1985. Last Accessed: April 11, 2011 at 21:09 EST.

[315] Jon Postel. RFC 760 - DoD Standard Internet Protocol. http://tools.ietf.org/html/rfc760, January 1980. Last Accessed: April 4, 2011 at 20:41 EST.

[316] Jon Postel. Internet Protocol DARPA Internet Program Protocol Specification. http://www.ietf.org/rfc/rfc791.txt, September 1981. Last Accessed: April 15, 2011 at 13:10 EST.

[317] Jennifer Preece, Yvonne Rogers, and Helen Sharp. *Interaction Design*. Wiley, John Wiley & Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, 2002.

[318] The Apache DB Project. Apache Derby. http://db.apache.org/derby/, December 2010. Last Accessed: April 15, 2011 at 16:00 EST.

[319] The Apache Software Project. Log Files - Apache HTTP Server. http://httpd.apache.org/docs/2.2/logs.html, 09 2010.

[320] The FreeBSD Project. The FreeBSD Project. http://www.freebsd.org/, January 1995-2011. Last Accessed: April 16, 2011 at 22:49 EST.

[321] Kevin Quiggle. MozManual. http://downloads.mozdev.org/mozmanual/en/mozmanual.pdf, 02 2004. Accessed on 24/07/2010.

[322] Dave Raggett, Arnaud Le Hors, and Ian Jacobs. HTML 4.01 Specification. http://www.w3.org/TR/html401/, December 1999. Last Accessed March 13, 2011 21:36 PM.

[323] Inc. Red Hat Middleware. Chapter 14. HQL: The Hibernate Query Language. http://docs.jboss.org/hibernate/core/3.3/reference/en/html/queryhql.html, January 2004. Last Accessed: April 16, 2011 at 19:02 EST.

[324] RENT-A-GURU. HTTP-ANALYZE - A Logfile Analyzer for Web Servers. http://http-analyze.org/index.php, January 2011. Last Accessed on January 4, 2011 20:41 EST.

[325] E. Rescorla. HTTP Over TLS. http://tools.ietf.org/html/rfc2818, May 2000. Last Accessed: April 4, 2011 at 19:13 EST.

[326] J. Reynolds. The Helminthiasis of the Internet. Informational 1135, Network Working Group, The Internet Society, University of Southern California Information Sciences Institute 4676 Admiralty Way Marina del Rey, CA 90292, December 1989. Last Accessed: April 11, 2011 at 20:45 EST.

[327] Allan Liska Rich Bowen, Daniel Lopez Ridruejo. *Apache Administrator's Handbook*. SAMS, Indianapolis, Indiana 46240 USA, March 2002.

[328] Ivan Ristic. ModSecurity: Open Source Web Application Firewall. http://www.modsecurity.org/projects/modsecurity/apache/, January 2004-2010. Last Accessed: April 4, 2011 at 16:36 EST.

[329] Ivan Ristic. *Apache security*. O'Reilly Media, Inc., pub-ORA-MEDIA:adr, 2005. Section 1.1.2,.

[330] Stephen Spainhour Robert Eckstein. *Webmaster in a Nutshell*. O'Reilly Media, 3rd edition, December 2002.

[331] Dr. Lawrence G. Roberts. The ARPANET & Computer Networks. http://www.packet.cc/files/arpanet-computernet.html, May 1995. Last Accessed: April 20, 2011 at 12:58 EST.

[332] D. Robinson and K. Coar. The Common Gateway Interface (CGI) Version 1.1. http://www.ietf.org/rfc/rfc3875.txt, October 2004. Last Accessed: April 6, 2011, at 10:54 EST.

[333] MK Rogers. Information Protection Association of Manitoba. In *Computer Forensics: Steps Toward Defining a Common Body of Knowledge*, 2002.

[334] RSnake. XSS (Cross site Scripting) Cheat Sheet Esp: for Filter Evasion. http://ha.ckers.org/xss.html, 1995-2009. Last Accessed on October 10, 2010.

[335] RSnake. Micro PHP LFI Backdoor Web Application Security Lab. http://ha.ckers.org/blog/20100128/micro-php-lfi-backdoor/, January 2010. Last Accessed on: March 13, 2011.

[336] Rusty Russell. Linux Ipchains - Howto. http://tldp.org/HOWTO/IPCHAINS-HOWTO.html, July 2000. Last Accessed: April 15, 2011 at 9:27 EST.

[337] Salvatore Sanfilippo. Visitors - Fast Web Log Analyzer. http://www.hping.org/visitors/, March 2009. Last Accessed on: January 28, 2011 at 10:49 EST.

[338] Joel Scambray and Mike Shema. *Hacking Exposed Web Applications*. McGraw-Hill / Osbourne, McGraw-Hill/Osborne 2600 Tenth Street Berkeley, California 94710 U.S.A., Jan 2002.

[339] Douglas Schweitzer. *Incident Response: Computer Forensics Toolkit*. Wiley Publishing Inc., Indianapolis, Indiana, USA, 2003. ISBN: 0-7645-2636-7.

[340] SecurityFocus. Internet Explorer 8 'toStaticHTML()' HTML Sanitization Bypass Weakness. http://www.securityfocus.com/bid/42467/info, 08 2010. Last Accessed on October 11, 2010.

[341] SecurityFocus. Opera Web Browser 10.62 and Prior Multiple Security Vulnerabilities. http://www.securityfocus.com/bid/43607/discuss, 09 2010. Last Accessed on October 11, 2010.

[342] United States Senate, editor. *United States Code: Federal Information Security Management Act*, volume S. 3474 of *Title 44*, 732 North Capitol Street, NW, Washington, DC 20401-0001, December 2008. Senate and House of Representatives of the United States of America in Congress, 110th Congress 2nd Session, United States Government Printing Office. Last Accessed: April 13, 2011 at 19:18 EST.

[343] Inc. Sendmail. Open Source - Sendmail.com. http://www.sendmail.com/sm/open_source/, January 1998-2011. Last Accessed: April 11, 2011 at 18:24 EST.

[344] Yakov Shafranovich. Common Format and MIME Type for Comma-Separated Values (CSV) Files. http://tools.ietf.org/html/rfc4180, October 2005. Last Accessed: April 13, 2011 at 12:40 EST.

[345] Barry Shaw. Bill 198 Sarbanes-Oxley Comes to Canada. http://www.itprojecttemplates.com/WP_SEC_BillC198.htm, January 2005. Last Accessed: April 13, 2011 at 17:00 EST.

[346] Jerry Shenk. Arcsight Logger Review. *A Sans Whitepaper*, 1:3, January 2009. Filename: loggerReview_Jan09.pdf Downloaded December 27, 2010.

[347] Sachin Shetty. Introduction to Spyware Loggers. http://www.symantec.com/connect/articles/introduction-spyware-keyloggers, April 2005. Last Accessed: April 10, 2011 at 10:06 EST.

[348] Jake Smith. The Spanner Collective Online Community. http://www.thespanner.co.uk/, 2010. better information to be gleaned from http://www.google.com/search?client=ubuntu&channel=fs&q=site2Fwww.thespanner.co.uk

[349] Randy Franklin Smith. *The Windows Server 2003 Security Log Revealed*. Monterey Technology Group Press, 2005-2007.

[350] Benjamin "snowhare" Franz. FTPWebLog 1.0.2. http://snowhare.com/utilities/ftpweblog/, January 2011. Last Accessed January 4, 2011 11:55 EST.

[351] Eugen J. Sobchenko. Oops! Proxy Server. http://oops-cache.org/about.html, February 2003. Last Accessed: April 12, 2011 at 8:57 EST.

[352] Edgewall Software. The Trac Project. http://trac.edgewall.org/, 2010. Last Accessed on November 11, 2010 @ 21:06 EST.

[353] Inc. Splunk. How Indexing Works. http://www.splunk.com/base/Documentation/latest/Admin/Howindexingworks, January 2011. Last Accessed on: Thu 27 Jan 2011 01:39:48 PM EST.

[354] Inc. Splunk. Splunk Overview. http://www.splunk.com/base/Documentation/latest/User/SplunkOverview, January 2011. Last Accessed on: Thu 27 Jan 2011 01:33:00 PM EST.

[355] Inverse Path S.r.l. Inverse Path - Research - Tenshi - Log Monitoring Tool. http://www.inversepath.com/tenshi.html, Jan 2010. Last Accessed on January 17, 2011 15:21 EST.

[356] Inverse Path S.r.l. Inverse Path - Research - Tenshi Manual Page. http://www.inversepath.com/tenshi_man.html, Jan 2010. Last Accessed on January 17, 2011 15:21 EST.

[357] Stefan Stapelberg. HTTP-ANALYZE 2.4 Online Manual. http://http-analyze.org/manual2.4/, August 2000. Last Accessed on January 4, 2011 20:37 EST.

[358] Bjarne Stroustrup. *The C++ Programming Language*. Addison - Wesley Professional, Corporate & Professional Publishing Group Addison-Wesley Publishing Company One Jacob Way Reading, Massachusetts 01867, 3rd edition, August 1997.

[359] Tony Stubblebine. *Regular Expression Pocket Reference*. O'Reilly Media, 2nd edition, July 2007.

[360] Dafydd Stuttard and Marcus Pinto. The web application hacker's handbook. Wiley Publishing, Inc., 2008. ISBN:978-0-470-17077-9 Web Tunnel Attacks: see pages 563 and 564.

[361] Ondrej Suchy. *Readme of Guard26*. underground.cz, http://www.penguin.cz/August 09. Contact E-Mail: ¡ondrej.suchy@underground.cz¿ Translation by Google translate from the README of Guard 2.6 Last Accessed at 16:22 EST December 22, 2010.

[362] Chris Sullo and David Lodge. Nikto2. http://cirt.net/nikto2, January 2010. Last Accessed: April 17, 2011 at 16:11 EST.

[363] David Swift. A Practical Application of SIM/SEM/SIEM Automatiing Threat Identification. Paper, SANS Infosec Reading Room, The SANS Institute 8120 Woodmont Avenue, Suite 205 Bethesda, Maryland 20814, December 2006. Last Accessed: April 11, 2011 at 9:58 EST.

[364] Cisco Systems. PIX/ASA as a DHCP Server and Client Configuration Example. http://www.cisco.com/en/US/products/hw/vpndevc/ps2030/products_configuration_example09186a00806c1cd5.shtml, October 2008. Last Accessed: April 15, 2011 at 11:56 EST.

[365] Talend. Open Source Data Integration Solutions for ETL. http://www.talend.com/solutions-data-integration/etl-for-analytics.php, January 2006-2011. Last Accessed: April 12, 2011 at 7:43 EST.

[366] Check Point Software Technologies. Firewall-1 Protection |Check Point Software. https://www.checkpoint.com/products/firewall-1/index.html, January 2010. Last Accessed: April 15, 2011 at 9:34 EST.

[367] Core Security Technologies. *Operational Documentation Core Wisdom*. CORE Labs Research and Development, Research and Development Center Humboldt 1967 1 Piso C1414CTU Buenos Aires Argentina, version 1.0 edition, August 2002. Last Accessed on December 31, 2010 at 13:05 EST, page 21.

[368] Brad Templeton. Reaction to the DEC Spam of 1978. http://www.templetons.com/brad/spamreact.html, May 2008. Last Accessed: March 15, 2011 16:34 PM.

[369] Kerry Thompson. LogSurfer & LogSurfer+ - Real Time Log Monitoring and Alerting. http://www.crypt.gen.nz/logsurfer/, January 2002. Last Accessed on Tue 18 Jan 2011 12:08:38 PM EST.

[370] Kerry Thompson. An Introduction to Logsurfer. http://www.crypt.gen.nz/papers/logsurfer.html, March 2004. Last Accessed on Thu 20 Jan 2011 10:54:14 PM EST.

[371] Erik Troan and Preston Brown. *Logrotate(8) - Linux Man Page*. Redhat, 1801 Varsity Drive Raleigh, North Carolina 27606 USA, 8 edition. Last Accessed on March 28, 2011, at 20:31 EST.

[372] Stephen Turner. Readme for Analog – Configuring the Output. http://www.analog.cx/docs/output.html#DESCRIPTIONS, December 2004. Last Accessed on December 25, at 21:37 EST.

[373] Stephen Turner. Analog: WWW logfile analysis. http://www.analog.cx/, June 2005. Last Accessed on December 25, at 21:37 EST.

[374] Risto Vaarandi. sec(1): simple event correlator - Linux man page. http://linux.die.net/man/1/sec. Last Accessed on: Tue 01 Feb 2011 02:57:40 PM EST.

[375] Risto Vaarandi. Index of /˜risto/sec/rulesets/. http://www.estpak.ee/~risto/sec/rulesets/, September 2009. Last Accessed: April 16, 2011 at 20:15 EST.

[376] Risto Vaarandi. SEC - Open Source and Platform Independent Event Correlation Tool. http://simple-evcorr.sourceforge.net/, January 2011. Last Accessed on: January 29, 2011 10:45 EST.

[377] Robert Vamosi. How Internet Explorer Could Drain Your Bank Account. http://reviews.cnet.com/4520-3513_7-5142439-1.html, July 2004. Last Accessed on November 9, 2010, 21:31 EST.

[378] Joost van Baal, Egon L. Willighagen, and Francis J. Lacoste. *Lire Developer's Manual*. Stichting LogReport Foundation, Stichting NLnet Science Park 140 Amsterdam, NL 1098XG, 2.0.99.1 edition, March 2008. Last Accessed: April 13, 2011 at 22:37 EST.

[379] Wietse Venema. The Postfix Home Page. http://www.postfix.org/, March 2011. Last Accessed: April 11, 2011 at 18:30 EST.

[380] Neil Fitzgerald James Edkins Annette Jonker Michael Voloshko. *Crystal Reports XI Official Guide*. Sams Publishing, United States of America, November 2007.

[381] W3C. Logging Control In W3C `httpd`. http://www.w3.org/Daemon/User/Config/Logging.html, July 1995. Last Accessed: April 5, 2011 at 19:08 EST.

[382] Peter Wainwright. *Pro Apache*. Apress - Springer-Verlag, New York, NY, USA, 3rd edition, 2004.

[383] Duane Wessels. *Web Caching*. O'Reilly Media, Sebastopol, CA USA 95472, 2001. Chapter 1, section 1.1.2.

[384] Inc. WhiteHat Security. WhiteHat Website Security Statistic Report. Statistic Report 9, WhiteHat Security, 3003 Bunker Hill Lane Santa Clara, CA 95054, Spring 2010. Last Accessed March 27, 2011, at 22:53 EST.

[385] Rainer Wichmann. The Samhain File Integrity / Host-Based Intrusion Detection System. http://www.la-samhna.de/samhain/index.html, January 2006. Last Accessed: February 23, 2011.

[386] Mark "Tarquin" Wilton-Jones. Javascript Tutorial - Using Cookies. http://www.howtocreate.co.uk/tutorials/javascript/cookies, September 2008. Last Accessed: March 24, 2011 11:06 EST.

[387] Dave Winer. RSS 2.0 Specification. http://www.rssboard.org/rss-specification, March 2009. Last Accessed: April 15, 2011 at 13:23 EST.

[388] European Research Consortium for Informatics World Wide Web Consortium (Massachusetts Institute of Technology and Keio University) Mathematics. Main Page. http://www.w3.org. All Rights Reserved. http://www.w3.org/Consortium/Legal/2002/copyright-documents-20021231 Last Accessed on October 10, 2010.

[389] Brian Wotring. *Host Integrity Monitoring using Osiris and Samhain*. Syngress Publishing Inc., Syngress Publishing, Inc. 800 Hingham Street Rockland, MA 02370, 2005.

[390] Brian Wotring and Bruce Potter. Osiris | Host Integrity Monitoring. http://osiris.shmoo.com/, January 2006. Last Accessed on February 22, 2011.

[391] Edward Z. Yang. HTML Purifier XSS Attacks Smoketest. http://htmlpurifier.org/live/smoketests/xssAttacks.php. Last Accessed on October 10, 2010.

[392] Daniel Yim. Brute Force in Algorithmic Analysis. http://cobweb.sfasu.edu/rball/342/Daniel_WebProject/#definition, November 2009. Last Accessed: April 28, 2011 at 12:06 EST.

[393] T. Ylonen and Ed. C. Lonvick. The Secure Shell (SSH) Authentication Protocol. http://tools.ietf.org/html/rfc4252, January 2006. Last Accessed: April 15, 2011 at 12:03 EST.

[394] Jonathan Zdziarski. *Apache Evasive Maneuvers Module For Apache 1.3 and 2.0 Version 1.10 README*. Copyright (c) Deep Logic, Inc., version 1.10 edition, 2005. email address: jonathan@nuclearelephant.com.