Web Application Forensics:

The Uncharted Territory

By Ory Segal, Sanctum Security Group

Introduction

"If criminals would always schedule their movements like railway trains, it would certainly be more convenient for all of us" (Sherlock Holmes – "The valley of fear")

Well, web application hackers don't schedule their movements – and they certainly don't use tools and attacks, that are easily spotted or investigated. Names such as: *Cross Site Scripting, Poison Null Byte and SQL Injection* are just the tip of the iceberg in a new war front.

IT professionals are used to monitoring network traffic, using network sniffers and state-of-the-art intrusion detection systems which analyze bizarre network behavior and watch patterns of weird packets considered as network attacks. Many companies also install tripwires and honey pots to catch hackers who are trying to hack in through defected telnet daemons, or FTP servers. But most people are overlooking what happens on the WWW and do not understand the traffic reaching the web application. For example, the following two entries in the web server's log look exactly the same:

[1] **2002-03-25 09:36:33 192.168.1.1 – 192.168.1.2 80 POST /script.asp – 200**

And

[2] **2002-03-25 09:36:34 192.168.1.1 – 192.168.1.2 80 POST /script.asp – 200**

It is almost impossible to see from this log file what the actual data sent to the web application was. Both entries may be valid requests or they may contain attacks to the web application that will lead to the exposure of your company's most important and vital information.

The need to have full insight into the activity surrounding your site, including who is coming to your site and what they are doing there, is more important than ever as more and more mission critical information and services are becoming web enabled. Installing firewalls, access control systems and encryption are all important security mechanisms – but futile when it comes to protecting the Web applications themselves.

This article will review the importance of having good Web application logging in place in order to conduct a forensics investigation. We'll look at the challenges and recommend a methodology for conducting an investigation, including techniques for evidence gathering. Finally, we will present the benefits of using Sanctum's Web application firewall, AppShield, as a forensics utility.

The Challenge

In order to discover and understand an attempted web application attack, we first need to gather all the clues from the crime scene. Collecting these "digital fingerprints" left by the reckless hacker requires that **all** of the following data fields are available, for every HTTP request:

- Date
- Time
- Client IP Address
- HTTP Method
- URI
- HTTP Query
- A Full Set of HTTP headers
- The full request body

Some of this data can be extracted from files such as the web server or application server log files, but unfortunately, the most crucial data is unavailable through these sources. Most web and application servers do not grant access to HTTP information such as the full set of HTTP headers and the request body. Without those fields many log entries look alike, and the person conducting the forensics will not be able to distinguish between valid requests and lethal web application attacks.

A simple example is the "invisible data in POST request" problem, mentioned in this paper's introduction. Let's take a look at a simple HTML form:

```
<FORM METHOD=POST ACTION="/cgi-bin/script.cgi">
<INPUT TYPE=HIDDEN NAME="template" VALUE="tempFile.html">
<INPUT TYPE=TEXT NAME="user" VALUE="enter username here">
<INPUT TYPE=PASSWORD NAME="pass" VALUE="">
<INPUT TYPE=SUBMIT VALUE="submit">
….
```

Now, let's take a look at the request made for this form through 3 different views. The first two are extracted from log files of Microsoft IIS/5.0 and Apache 1.3.24, and the last one will be the actual request:

[**Microsoft IIS/5.0 log file**]:

```
2002-05-05 13:24:45 192.168.1.1 – 192.168.1.2 80 POST /cgi-bin/script.pl - 200 381
322 192.168.1.2 Mozilla/4.7+[en]+(WinNT;+I) - -
```

[**Apache log file**]:

```
192.168.1.1 - - [05/May/2002:16:21:53 +0300] "POST /cgi-bin/script.pl HTTP/1.0" 200
150
```

[**Actual request]:**

```
POST /cgi-bin/script.pl HTTP/1.0
Host: localhost
User-Agent: Mozilla/4.7 [en] (WinNT; I)
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip
Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Content-length: 60

template=../../../../../boot.ini%00.html&user=test&pass=test
```

As seen, the most important data does not exist in the web servers' logs. Any attempt to conduct a web application forensics investigation on this site, will surely fail to recognize that this request is a "Poison null byte" attack, which successfully retrieves the "boot.ini" file.

Our next example demonstrates the lack of HTTP headers logging, which is another problem that exists in web server log files. This example involves a famous Microsoft IIS/5.0 attack. Specifically, the "Translate: f" source code disclosure attack. By appending a "\" character to a request for a server side script, and adding an HTTP header with the value "Translate: f", any attacker can retrieve the source code of server side scripts, such as ASP scripts. Obtaining the source code of server side scripts grants the attacker deeper knowledge of the logic behind the web application. This knowledge helps the attacker to develop further attacks, which are by far more dangerous.

[**Actual request]**

```
GET /login.asp\ HTTP/1.0
Host: 192.168.1.2:80
Accept: */*
Accept-Language: en-us
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT 5.0)
Content-Type: application/x-www-form-urlencoded
Translate: f
```

[**Microsoft IIS/5.0 log file**]:

```
2002-05-05 13:43:55 192.168.1.1-192.168.1.2 80 GET /login.asp\ - 200 179 212
192.168.1.2:80 Mozilla/4.0+(compatible;+MSIE+5.5;+Windows+NT+5.0) - -
```

In this case, the log file created by the Microsoft IIS server does not include the complete set of HTTP headers. Thus, the attack looks just like a simple mistake in a perfectly valid HTTP request.

The above examples are just drops in a sea of problems which auditors face when trying to extract important information needed to conduct a web application forensics investigation. We will examine solutions for these problems later in this paper.

Web Application Forensics – Methodology and Practice

"Following a standard methodology is crucial to successful and effective computer forensics. Just as professional programmers use a thorough programming methodology, computer forensic professionals should use a thorough investigative methodology" (Jim McMillian, "Importance of a Standard Methodology in Computer Forensics." May 2000)

In order to conduct a thorough analysis of the hacking attempt, it is suggested that the investigator follow the *methodology* steps outlined here:

1. Protect the web application (could be several servers) during forensics examination from any possible alteration or data corruption.
2. Discover all files needed for the forensics investigation. This includes:
    a. Web server(s) and application server(s) logs
    b. Server side scripts which are used by the web application
    c. Web server(s) and application server(s) configuration files
    d. Any 3rd. party installed software logs and vital files.
    e. Operating system logs and vital system files

    Remember that the files may be spread over several computer systems, which together comprise the web application.
3. Analyze the collected data; try to create an exact chain of events. (techniques will be explained later)
4. Summarize findings, and make a log of all files and data extracted from the web application.

Once this is understood, the following *practices* should be followed to ensure a comprehensive forensics procedure can take place:

- Log file protection: As we have seen earlier in this paper, web application forensics is based mainly on log files and the integrity of the data they contain. It is highly recommended that you **protect your log files,** as any serious hacker will try to manipulate them. There are several ways to protect the logs, such as:

  a. Setting the proper permissions to the log files

  b. Keeping the log files out of the hacker's reach. This can be done by using some sort of back up utility, which will save the log files on a remote server. (The back up routine should run in short time intervals)

  c. Using some sort of checksum in order to verify the log files integrity

- Collecting the evidence: If your web or application server does not include the important log file entry fields needed for conducting the forensics investigation, you should use a third party utility which does. We will see in the last section of this paper, how Sanctum's AppShield logging facilities can be used to retrieve this information easily and efficiently.

- Divide and Conquer: During the investigation, try to divide the log file according to user sessions, e.g. if you are using some sort of a session token, for example a cookie, try to group log entries according to the token. The grouping will give you better understanding of the session flow and timeline, and will remove noise created by other users in the log file. After the grouping is done, you are left with clusters of requests grouped according to the user or the originating IP address. Each cluster is organized internally according to the time that the request was made. The organized cluster describes the "User session flow".

- Digital fingerprints powder: Many attacks use 'suspicious characters', also referred to as 'hazardous characters', which are easily spotted in the log files by the human eye, and sometimes by intrusion detection systems. Hackers often use anti-IDS techniques in order to cover the tracks made by the attacks. These techniques involve encoding of the suspicious characters using URL-encoding, or other types of encoding, which are supported by the web or application server (such as overlong UTF-8 – Unicode). It is highly recommended that you use some sort of decoding script ('digital fingerprints powder') in order to help you with the reading of the log file. But, you must remember that you should not decode unreadable characters such as: \t \n \0 and so on. They are much more easily read in their encoded form – e.g. %09, %0d%0a and %00 respectively.

- Intended application flow: Although it should not be considered reliable, the HTTP "Referer" header may sometimes help to verify that a user was following the intended application flow. For example, if the first page in the web application is /login.asp and right after it, the user is directed to the /account.asp page, when you see a request in the log file for /account.asp with the HTTP "Referer" header set to something else than "/login.asp" – you should be alerted.

Note: the HTTP "Referer" header should not be used as a security measure! A request lacking a "Referer" or containing a bogus one should alert your attention. A request containing a 'valid' value *should not* be considered safe, because HTTP headers are easily spoofed.

- Finding the digital fingerprints: several articles have been written which describe the fingerprints and patterns left by web application hacking attempts. For example, "*Fingerprinting port 80 attacks: a look into web server, and web application attack signatures*" By Zenomorph http://www.cgisecurity.com/papers/fingerprint-port80.txt. It is out of the scope of this paper to try and name all of the attack patterns, but it is very easy to form a basic thumb rule, which describes most of them.

  When looking in the log files, any of the following patterns should alert your attention:

  - Special characters (E.g. Meta-Characters such as null byte, pipe '|' and any string terminating characters such as quote or apostrophe)
  - Bizarre or irregular HTTP Methods (E.g. PROPFIND, HEAD, PUT)
  - Binary utilities (E.g. cmd.exe in Win32 systems, or /bin/ls in Unix systems)
  - Vital system files (E.g. the SAM file in Win32 systems, or /etc/passwd in Unix systems)
  - Any files outside the web application directory
  - Any files outside the virtual web server root
  - Rapid/Massive requests to CGI scripts. This is usually the case when hackers use CGI scanning automated utilities.
  - Extremely long requests (E.g. buffer overflow attempts – GET /AAAAAAA……)

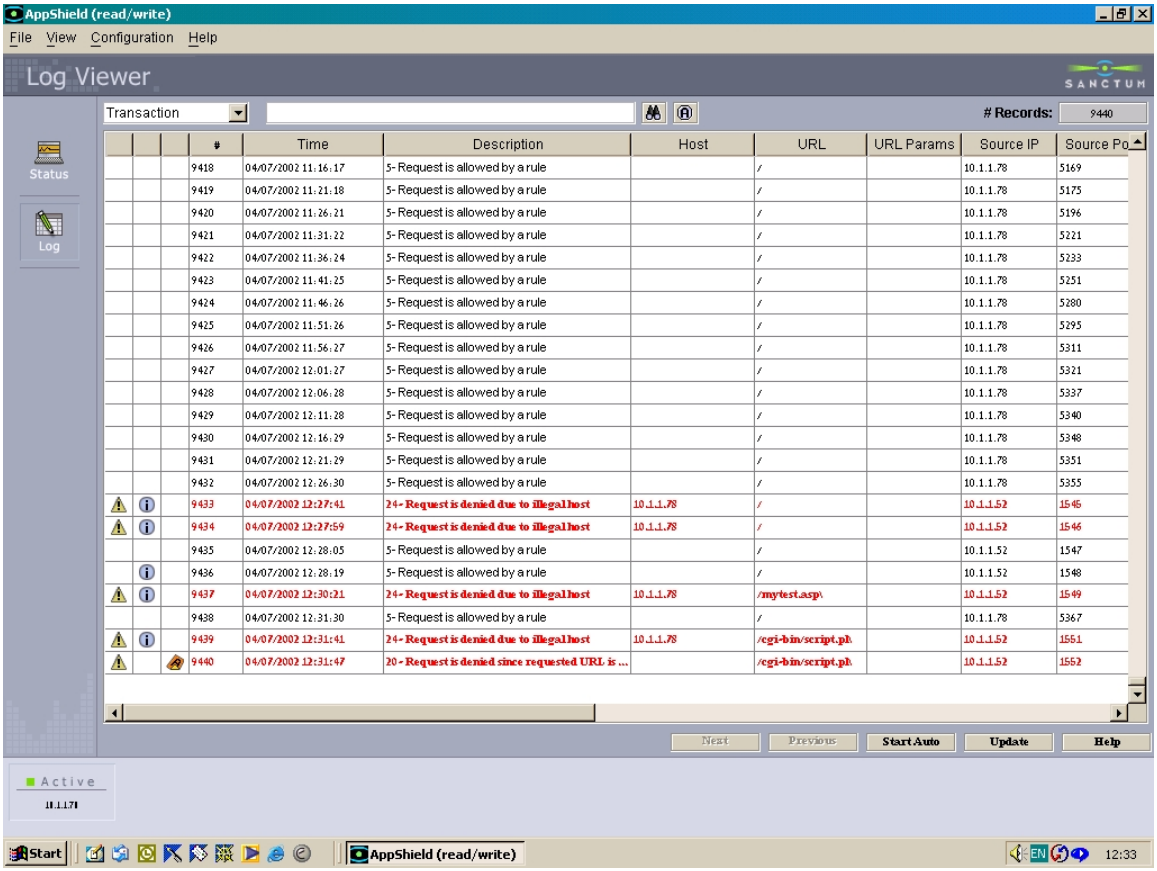
Using AppShield for Web Application Forensics

Sanctum's AppShield software is a Web Application Firewall built on a security proxy architecture that enforces a positive security model blocking any type of application manipulation. It is an active system that monitors and responds to any unusual or unauthorized behavior anywhere within a site, blocking attacks before they can reach the site. The web application security provided by AppShield ensures that Web applications (both the application and web logic) can only be used the way they were intended by the developer (i.e. intended application flow). Any attempt at manipulating them is directly blocked, preventing the unauthorized use of an e-Business' resources or customer information. Even un-patched and insecure systems are made secure by AppShield allowing site owners to perform security patching with regularly scheduled maintenance. Most importantly, AppShield does not require any patches or signature files to stay up to date (i.e. you do not need to update AppShield with new patterns of

attacks), so the unpredictable and costly cycle of security vulnerabilities and patching is broken once and for all.

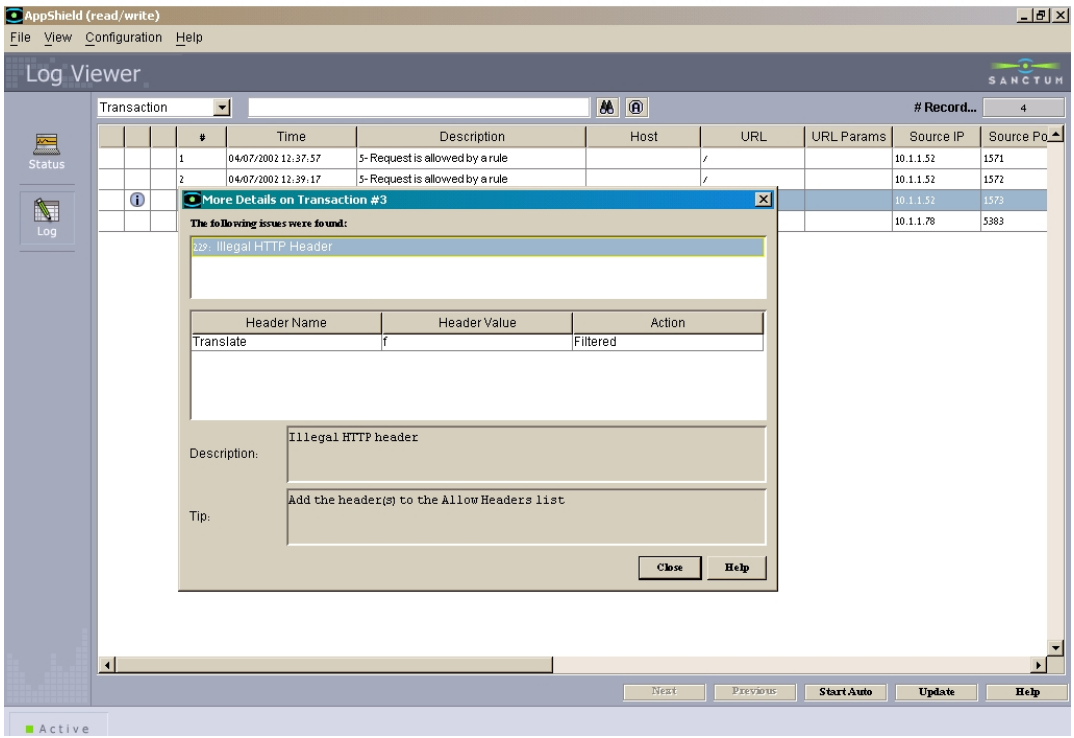
AppShield includes many logging features which add a great value to web application forensics. Here are some examples:

1. AppShield's **Log Viewer** is easy to use. Any attempt to manipulate the web application is easily found in the logs –when you get to the crime scene just request the evidence that AppShield has already collected for you. The data can be presented and analyzed in a variety of formats based on the investigator or auditor's preference and requirements.



2. AppShield employs Sanctum's unique, patented Dynamic Policy Recognition Engine (DPRE) technology to examine and enforce application security policies in real time with the most powerful HTTP security proxy ever developed. This type of positive policy does not require a negative signature database, is unique to each user session, is always accurate and up to date, and requires minimal administration. Because of AppShield's Dynamic Policy Recognition Engine, it is ensured that web applications will only be used the way they were intended by the developer (i.e. Intended application flow). This feature renders all CGI scanning and attacks (as smart as they will be) useless, because the vulnerable CGIs do not belong to the web application.

3. In case a hacker tries to send some HTTP header, which is illegal, the Log Viewer will present it clearly, unlike most web servers' log files. The following page contains a screenshot of a request which included an illegal HTTP header (Translate: f)



4. Several attacks use anti-IDS tactics in order to corrupt the log files by injecting hazardous unreadable characters, such as backspace or carriage return. These attacks will not affect AppShield's logging facilities.

5. AppShield includes a "**TCP/IP packet logging**" utility, which logs every HTTP related packet. This utility allows you to debug the web and application server's HTTP behavior, and may help with understanding specific attacks and hacking attempts. This tool is just like a detective's magnifying glass; you can't get any closer than this!

6. When using AppShield's log viewer, both POST and GET requests, or any other HTTP method for that matter, are fully logged and analyzed and can be easily found by anyone.

7. Last but not least, many web application attacks look like valid requests and do not have a specific fingerprint such as changing a price parameter's value from $199.99 to $1.99. These attacks most likely will never be caught by IDS systems or the human eye, because they do not match any "known pattern". This is where AppShield's power is most valued. AppShield will block and alert any kind of web application attack, without the need of a pattern or a "fingerprint", by using the Dynamic Policy Recognition Engine (DPRE).

Conclusion

Conducting web application forensics is heavily based on the assumption that all HTTP data is kept in the log files, and is easily accessed when needed. Sadly, many contemporary web and application servers do not include proper handling of HTTP communications logging. Those that do, present the user with difficulties when trying to extract the data in a manner that will help to conduct a proper investigation of a hacking attempt (or even worse- a hacking attempt that succeeded!).

In order to implement the proper incident response procedures in your organization and to use the standard methodology for web application forensics, it is advised that you use a product such as Sanctum's AppShield, which is tailored specifically for this task and offers the comprehensive logging facilities, analysis and protection needed for today's forensic analysis.

References and relevant links

- Sanctum Inc. web site: http://www.sanctuminc.com
- Computer Forensics – An Overview: http://rr.sans.org/incident/forensics.php
- "Fingerprinting port 80 attacks: a look into web server, and web application attack signatures" Zenomorph: http://www.cgisecurity.com/papers/fingerprint-port80.txt.
- Farmer, Dan and Venema, Wietse. "Forensic Computer Analysis: An Introduction." Dr. Dobb's Journal. September, 2000. http://www.ddj.com/documents/s=881/ddj0009f/0009f.htm
- McMillian, Jim. "Importance of a Standard Methodology in Computer Forensics." May 2000. http://www.sans.org/infosecFAQ/incident/methodology.htm
- CodeRed II: Incident Handling Process and Procedures http://rr.sans.org/incident/code_red_II.php