Tivoli. software

IBM

IBM Tivoli Access Manager for e-business: Tracing HTTP Connections

White Paper

Ori Pomerantz (orip@us.ibm.com)

June 2009

Table of Contents

Introduction

About this Paper

Problems related to Tivoli Access Manager for e-business often require tracing of HTTP connections. In this paper you will learn how to obtain such traces~~,~~ and how to interpret them.

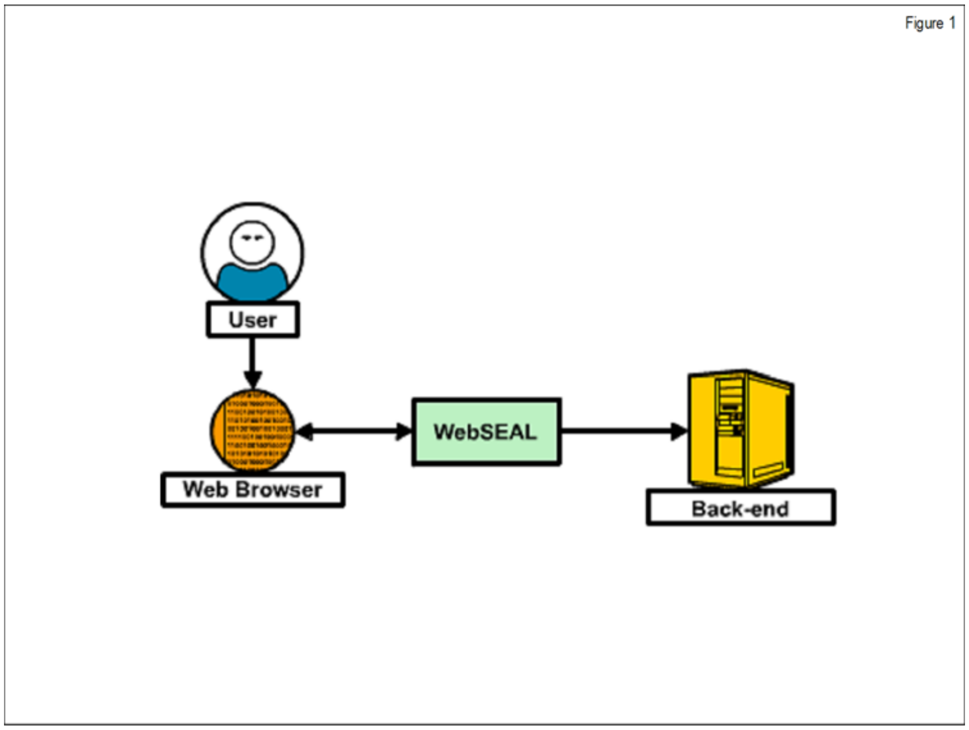
Audience

This paper will help implementors and system administrators who need to solve problems with WebSEAL.

White Paper

1 Why Do It?

WebSEAL is an HTTP proxy that receives an HTTP connection from a browser. If an action is authorized, WebSEAL opens a separate HTTP connection to the back-end server.



WebSEAL does not just transfer the information from one connection to the other. It also modifies the URL to interpret junctions. It adds information to the HTTP header for the back-end connection and cookies for the front-end connection. Those changes might cause problems. The easiest way to trace them is to look at the HTTP connections.

2 HTTP Header Traces

WebSEAL always changes the HTTP headers. You can identify many problems by examining the traces for those HTTP headers.

2.1 Creating HTTP Header Traces

To create a trace of HTTP headers, run the following **pdadmin** command to start tracing:

```
server task <WebSEAL instance> trace set pdweb.debug 2 \
      file path=<file path>
```

The trace will appear in the path you specified.

Run the following **pdadmin** command to stop the trace:

```
server task <WebSEAL instance> trace set pdweb.debug 0
```

2.2 Interpreting HTTP Header Traces

The HTTP header trace contains all of the HTTP headers and other information. This white paper only documents the lines that are useful for WebSEAL diagnostic purposes. For information about other fields, see the HTTP specifications in RFC 2616, available at the following URL:

http://www.w3.org/Protocols/rfc2616/rfc2616.html

2.2.1 Request from the Browser

In this section is an initial request from a browser to WebSEAL. This text is written all on one line.

```
2009-04-28-19:48:37.622-05:00I----- thread(3)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:134: ----------------
- Browser ===> PD -----------------
```

The relevant parts of the line are the time stamp at the beginning of the line and the arrow indicating that this is a request coming from the browser to the Policy Director (**PD)**. Policy Director is the former name for Tivoli Access Manager for e-business; therefore, this is a request going from a browser to WebSEAL.

The ID for the WebSEAL thread that handled the request is followed by the HTTP request sent by the browser.

```
Thread_ID:2975017888
GET /junc/index.html HTTP/1.1
```

The first line of the HTTP request is the request itself. In this case, the browser is requesting **GET** the file **/junc/index.html** using version **1.1** of the **HTTP** protocol.

The request is followed by the HTTP header fields. Some of these fields are optional, and they can appear in any order. The following line is the host part of the URL (and the port number, if specified):

```
host: tam
```

The host name is useful in the case of virtual host junctions, where this field determines the junction and the back-end server. The following line is the identity of the browser that sent the request:

```
user-agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.1.10) Gecko/20071015 SUSE/2.0.0.10-0.11 Firefox/
2.0.0.10
```

The following line specifies the end of the entry, in this case the request from the browser.

```
---------------------------------------------------
```

2.2.2 Request for Basic Authentication

The request in the previous section came from an unauthenticated browser. In the next section WebSEAL requests basic authentication before proceeding any further. This code is also on one line.

```
2009-04-28-19:48:37.670-05:00I----- thread(3)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:178: ----------------
- Browser <=== PD ----------------
```

Notice that the arrow now points from PD to the Web browser. This is a response.

The first line of the HTTP reply is the reply itself.

```
HTTP/1.1 401 Unauthorized
```

In this case, the server is using version **1.1** of the **HTTP** protocol, the same version as the browser. The response is **401**, meaning that the request requires authentication. This reply happens in response to a request that is not authenticated~~,~~ or in response to a response authenticated as an unauthorized user. The number is followed by a string, which can be displayed to the user.

The following field specifies the Platform of Privacy Preferences settings, a standard mechanism to specify the privacy practices of Web sites.

```
p3p: CP=”NON CUR OTPi OUR NOR UNI”
```

You can read more about P3P at the following URL:

http://www.w3.org/P3P

The following field identifies the Web server, in this case WebSEAL.

```
server: WebSEAL/6.1.0.0 (Build 080319)
```

Servers that ask for basic authentication include the following field so that the browser can show it to the user.

```
www-authenticate: Basic realm=”Access Manager for e-
business”
```

To modify this value, change the **basic-auth-realm** property in the **[ba]** stanza of the instance configuration file.

```
Set-Cookie: PD-S-SESSION-
ID=2_mNCraP8I1j7MvXEqWXfaxFbawImnvMOwKHsrU3MF5mqJpnqS;
Path=/; Secure
```

This code is all on one line and sets a cookie called **PD-S-SESSION-ID** that stores the session ID. To modify the cookie name, change the **ssl-session-cookie-name** (or, for HTTP, **tcp-session-cookie-name**) in the **[session]** section of the instance configuration file.

The information after the semicolon (;) is the cookie attributes. The **Path** attribute specifies that this cookie should be sent back with any request that starts with a slash (/), which means any request. The **Secure** attribute specifies that the cookie needs to be secure, for example, not to be sent back in clear text. For more information about cookies, see the draft specifications that are available at the following URL:

http://web.archive.org/web/20060424004149/wp.netscape.com/newsref/std/cookie_spec.html.

2.2.3 Basic Authentication Response

The basic authentication response is almost identical to the original request, but with two additional lines.

The **authorization:** field specifies the user name and password. Because it is sensitive information, the value is not shown in the trace.

```
authorization:***********************************
```

The following line returns the session ID cookie set by the server in the previous section. If you have to use a trace from a production environment, you can use the session ID to identify which requests come from which browser.

```
Cookie: PD-S-SESSION-
ID=2_mNCraP8I1j7MvXEqWXfaxFbawImnvMOwKHsrU3MF5mqJpnqS
```

2.2.4 Request to the Back End

WebSEAL decided that the request from the browser was authorized. The next entry is the request that WebSEAL sends to the back-end server.

```
2009-04-28-19:48:45.977-05:00I----- thread(4)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:134: ----------------
- PD ===> BackEnd ----------------
```

This text is all on one line and represents the request that goes from PD (WebSEAL) to the back-end server.

This is the HTTP request:

```
GET /index.html HTTP/1.1
host: backend:81
```

The host field is taken from the junction. Notice that it includes the port number when that is not the default (80 for HTTP, 443 for HTTPS).

The user agent field is relayed from the browser. This way, the back-end server can send different replies to different browsers.

```
user-agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.1.10) Gecko/20071015 SUSE/2.0.0.10-0.11 Firefox/
2.0.0.10
```

The **via:** field contains proxy information. WebSEAL is a proxy, so the information for WebSEAL is listed here.

```
via: HTTP/1.1 tam:443
```

This is one of a number of HTTP header fields that could be inserted by WebSEAL when communicating with the back end. It is specified in the **server_name** property of the **[header-names]** stanza of the instance configuration file.

```
iv_server_name: default-webseald-tam.ibm.com
```

If the junction had been created with the **-c** parameter, you would see the user identification fields: **iv-user**, **iv-user-l**, **iv-groups**, and **iv-creds**.

2.2.5 Back End Response

The back-end server responds to the request. The following text, all on one line, shows that the response goes from the back-end server to PD (WebSEAL).

```
2009-04-28-19:48:46.102-05:00I----- thread(4)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:178: ----------------
- PD <=== BackEnd -----------------
```

The following field identifies the server, which can be useful in verifying that junctions direct to the correct back-end server.

```
server: IBM_HTTP_Server
```

2.2.6 Response to the Browser

Finally, the browser gets the page the user requested. The text, all on one line, shows that the response goes from PD (WebSEAL) to the browser.

```
2009-04-28-19:48:46.102-05:00I----- thread(4)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:178: ----------------
- Browser <=== PD -----------------
```

WebSEAL sets a new session ID value.

```
Set-Cookie: PD-S-SESSION-
ID=2_NZm8soRZG9QP7BDkktl1RXlPmlj0NPLmk-9Q1ubddvU3lGCh;
Path=/; Secure
```

2.2.7 Browser Request for Additional Information

Displaying an HTML page often requires the browser to retrieve additional files: images, style sheets, and so on. In the following entry you can see the Web page that caused the request in the **referer:** field.

```
2009-04-28-19:48:49.895-05:00I----- thread(5)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:134: ----------------
- Browser ===> PD ----------------
Thread_ID:2973952928
GET /junc/http_server_styles.css HTTP/1.1
accept: text/css,*/*;q=0.1
accept-encoding: gzip,deflate
accept-language: en-us,en;q=0.5
authorization:***********************************
connection: keep-alive
host: tam
keep-alive: 300
referer: https://tam/junc/index.html
user-agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
rv:1.8.1.10) Gecko/20071015 SUSE/2.0.0.10-0.11 Firefox/
2.0.0.10
accept-charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Cookie: PD-S-SESSION-ID=2_NZm8soRZG9QP7BDkktl1RXlPmlj0NPLmk-
9Q1ubddvU3lGCh
```

2.3 Using Header Traces

The following example is a problem that can be solved using a header trace. WebSEAL is set up for forms authentication. A user complains that WebSEAL ignores the authentication. The relevant part of the trace is:

```
2009-04-30-21:07:21.712-05:00I----- thread(17)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:134: ----------------
- Browser ===> PD ---------------- Thread_ID:2961496992
GET /junc HTTP/1.1
accept: text/html, image/jpeg, image/png, text/*, image/*,
*/*
accept-encoding: x-gzip, x-deflate, gzip, deflate
accept-language: en
connection: Keep-Alive
host: 192.168.90.3
user-agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux
2.6.19; X11) KHTML/3.5.5 (like Gecko) (Debian package
4:3.5.5a.dfsg.1-5)
accept-charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
cache-control: no-cache
pragma: no-cache
```

The browser is asking for a page protected by WebSEAL.

```
---------------------------------------------------
 2009-04-30-21:07:21.713-05:00I----- thread(17)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:178: ----------------
- Browser <=== PD ---------------- Thread_ID:2961496992
HTTP/1.1 200 OK
content-length: 1970
content-type: text/html
date: Fri, 01 May 2009 02:07:21 GMT
p3p: CP="NON CUR OTPi OUR NOR UNI"
server: WebSEAL/6.1.0.0 (Build 080319)
pragma: no-cache
cache-control: no-cache
Set-Cookie: PD-H-SESSION-
ID=4_ITXlIcoIVJkyRq8fpH3BXYmKW792wrlKZHBEkq8NLV57109J;
Path=/
```

WebSEAL replies, probably with the log-on form. It also attempts to set a cookie, **PD-H-SESSION-ID**.

```
---------------------------------------------------
 2009-04-30-21:07:26.718-05:00I----- thread(17)
trace.pdweb.debug:1 /project/amweb610/build/amweb610/src/
pdweb/webseald/http/server/handle-call.cpp:139: IO error
during request parsing
2009-04-30-21:07:30.168-05:00I----- thread(18)
trace.pdweb.debug:2 /project/amweb610/build/amweb610/src/
pdweb/webseald/ras/trace/debug_log.cpp:134: ----------------
- Browser ===> PD ---------------- Thread_ID:2960964512
POST /pkmslogin.form HTTP/1.1
accept: text/html, image/jpeg, image/png, text/*, image/*,
*/*
accept-encoding: x-gzip, x-deflate, gzip, deflate
accept-language: en
connection: Keep-Alive
content-length: 57
content-type: application/x-www-form-urlencoded
host: 192.168.90.3
referer: http://192.168.90.3/junc
user-agent: Mozilla/5.0 (compatible; Konqueror/3.5; Linux
2.6.19; X11) KHTML/3.5.5 (like Gecko) (Debian package
4:3.5.5a.dfsg.1-5)
accept-charset: iso-8859-1, utf-8;q=0.5, *;q=0.5
cache-control: no-cache
pragma: no-cache
```

The browser posts to **/pkmslogin.form**, presumably the logon information. However, it does not return the cookie. The problem might be that the cookies are turned off in the browser. Forms authentication cannot work without cookies.

3 Message Body Traces

To solve some problems, you need to look at the message bodies, the actual files transferred using HTTP. To do that, use **pdweb.snoop**. It provides the same function as a network sniffer, but provides it within WebSEAL outside the SSL tunnel so that the encryption is not an issue.

3.1 Creating Message Body Traces

To create a trace that includes the message body, run the following **pdadmin** command to start tracing:

```
server task <WebSEAL instance> trace set pdweb.snoop 9\
      file path=<file path>
```

The trace will appear in the path you specified.

Run the following **pdadmin** command to stop the trace:

```
server task <WebSEAL instance> trace set pdweb.snoop 0
```

Instead of **pdweb.snoop**, you can use **pdweb.snoop.client** to trace only the communication between WebSEAL and the client. You can also use **pdweb.snoop.jct** to trace only the communication between WebSEAL and the back-end server.

3.2 Interpreting Message Body Traces

The following example is the output of **pdweb.snoop**.

```
2009-04-29-23:25:49.966-05:00I----- thread(16)
trace.pdweb.snoop.jct:1 /project/amwebrte610/build/
amwebrte610/src/pdwebrte/webcore/amw_snoop.cpp:170:
```

The header line shows the time and date. It also shows the exact component that generated the message, either **pdweb.snoop.client** for communication between a browser and WebSEAL or **pdweb.snoop.jct** for communication between WebSEAL and a back-end server.

The following line shows the IP address and port number for the local (WebSEAL) and remote (browser or back-end server).

```
----------------------------------------
Thread 2963569568; fd 18; local 192.168.90.3:54192; remote
192.168.90.4:81
Receiving 454 bytes
```

The main snoop output consists of three columns: byte number (in hexadecimal), the bytes (also in hexadecimal), and the bytes as ASCII text. The third column is typically the most useful one.

```
0x0000   4854 5450 2f31 2e31 2034 3034 204e 6f74        HTTP/1.1.404.Not
0x0010   2046 6f75 6e64 0d0a 4461 7465 3a20 5765        .Found..Date:.We
0x0020   642c 2032 3920 4170 7220 3230 3039 2031        d,.29.Apr.2009.1
0x0030   393a 3238 3a30 3020 474d 540d 0a53 6572        9:28:00.GMT..Ser
0x0040   7665 723a 2049 424d 5f48 5454 505f 5365        ver:.IBM_HTTP_Se
0x0050   7276 6572 0d0a 436f 6e74 656e 742d 4c65        rver..Content-Le
0x0060   6e67 7468 3a20 3238 310d 0a43 6f6e 6e65        ngth:.281..Conne
0x0070   6374 696f 6e3a 2063 6c6f 7365 0d0a 436f        ction:.close..Co
0x0080   6e74 656e 742d 5479 7065 3a20 7465 7874        ntent-Type:.text
0x0090   2f68 746d 6c3b 2063 6861 7273 6574 3d69        /html;.charset=i
0x00a0   736f 2d38 3835 392d 310d 0a0d 0a3c 2144        so-8859-1....<!D
```

The lines show an HTTP header (the error message **404**, which means a file requested from the server was not found). To preserve the column format, nonprintable characters, such as new line, are printed as dots. You can identify them by looking at the bytes in hexadecimal. The characters, **0d0a**, would be in the location corresponding to a new line.

Notice the two consecutive new lines (shown in bold). They mark the end of the header and the beginning of the message body.

```
0x00b0   4f43 5459 5045 2048 544d 4c20 5055 424c        OCTYPE.HTML.PUBL
0x00c0   4943 2022 2d2f 2f49 4554 462f 2f44 5444        IC.”-//IETF//DTD
0x00d0   2048 544d 4c20 322e 302f 2f45 4e22 3e0a        .HTML.2.0//EN”>.
0x00e0   3c68 746d 6c3e 3c68 6561 643e 0a3c 7469        <html><head>.<ti
0x00f0   746c 653e 3430 3420 4e6f 7420 466f 756e        tle>404.Not.Foun
0x0100   643c 2f74 6974 6c65 3e0a 3c2f 6865 6164        d</title>.</head
0x0110   3e3c 626f 6479 3e0a 3c68 313e 4e6f 7420        ><body>.<h1>Not.
0x0120   466f 756e 643c 2f68 313e 0a3c 703e 5468        Found</h1>.<p>Th
0x0130   6520 7265 7175 6573 7465 6420 5552 4c20        e.requested.URL.
0x0140   2f69 6d61 6765 732f 6f64 6f74 2e6a 7067        /images/odot.jpg
0x0150   2077 6173 206e 6f74 2066 6f75 6e64 206f        .was.not.found.o
0x0160   6e20 7468 6973 2073 6572 7665 722e 3c2f        n.this.server.</
0x0170   703e 0a3c 6872 202f 3e0a 3c61 6464 7265        p>.<hr./>.<addre
0x0180   7373 3e49 424d 5f48 5454 505f 5365 7276        ss>IBM_HTTP_Serv
0x0190   6572 2053 6572 7665 7220 6174 2062 6163        er.Server.at.bac
0x01a0   6b65 6e64 2050 6f72 7420 3831 3c2f 6164        kend.Port.81</ad
0x01b0   6472 6573 733e 0a3c 2f62 6f64 793e 3c2f        dress>.</body></
0x01c0   6874 6d6c 3e0a                                 html>.
```

These lines show the HTML, which is the HTTP message body. To make the HTML easier to read, put the lines together and translate the control characters, as listed in the following table.

Hexadecimal Byte	Character
09	Tab
0a	New line
0d	Ignore
20	Space

By translating **0a** to a new line and **20** to a space, you can make the HTML easier to read.

```
<!DOCTYPE HTML PUBLIC “-//IEFT//DTD HTML 2.0//EN”>
<html><head>
<title>404 Not Found</title>
</head><body>
<h1>Not Found</h1>
<o>The requested URL /images/odot.jpg was not found on this
server.</p>
<hr />
<address>IBM_HTTP_Server Server at backend Port 81</address>
</body></html>
```

3.3 Using Message Body Traces

The following example is a problem that can be solved by tracing the message body. A Web page contains a link that does not work through WebSEAL. This is the relevant part of the trace:

```
2009-04-30-22:58:13.895-05:00I----- thread(24) trace.pdweb.snoop.client:1 /
project/amwebrte610/build/amwebrte610/src/pdwebrte/webcore/
amw_snoop.cpp:196:
----------------------------------------
Thread 2972412832; fd 8; local [::ffff:192.168.90.3]:443; remote
[::ffff:192.168.90.3]:56700
Sending 3727 bytes
0x0000   4854 5450 2f31 2e31 2032 3030 204f 4b0d        HTTP/1.1.200.OK.
0x0010   0a61 6363 6570 742d 7261 6e67 6573 3a20        .accept-ranges:.
0x0020   6279 7465 730d 0a63 6f6e 7465 6e74 2d74        bytes..content-t
0x0030   7970 653a 2074 6578 742f 6874 6d6c 0d0a        ype:.text/html..
.
.
.
0x0450   3c48 323e 5765 6c63 6f6d 6520 746f 2074        <H2>Welcome.to.t
0x0460   6865 2042 6163 6b65 6e64 2053 6572 7665        he.Backend.Serve
0x0470   723c 2f48 323e 0a0a 3c73 6372 6970 743e        r</H2>..<script>
0x0480   0a64 6f63 756d 656e 742e 7772 6974 6528        .document.write(
0x0490   223c 6120 6872 6566 3d5c 2268 7474 703a        “<a.href=\”http:
```

```
0x04a0   2f2f 6261 636b 656e 643a 3831 2f6c 696e        //backend:81/lin
0x04b0   6b65 642e 6874 6d6c 5c22 3e43 6c69 636b        ked.html\”>Click
0x04c0   2068 6572 652e 3c2f 613e 3c2f 6272 3e22        .here.</a></br>”
0x04d0   293b 0a3c 2f73 6372 6970 743e 0a0a 3c74        );.</script>..<t
0x04e0   723e 0a0a 2020 2020 2020 2020 3c74 643e        r>..........<td>
```

As you can see, there is an absolute link inside a script that is not translated. To fix this problem, recreate the junction with **-j**.

Conclusion

Summary

You should now be able to view the information passing through WebSEAL and use it to troubleshoot HTTP-related problems.

Resources

- The HTTP protocol is defined in RFC 2616, available at the following URL:

  http://tools.ietf.org/html/rfc2616

- Cookies are explained in the draft specifications, available at the following URL:

  http://web.archive.org/web/20060424004149/wp.netscape.com/newsref/std/cookie_spec.html

- An explanation of P3P is available at the following URL:

  http://www.w3.org/P3P