T2-11 Trace File Analysis - Analyzing HTTP Traffic Behavior
April 2, 2008

Tony Fortunato
Sr Network Specialist | The Technology Firm

SHARKFEST '08
Foothill College
March 31 - April 2, 2008
About your Presenter

Tony Fortunato, Sr Network Specialist, The Technology Firm

Certified Fluke Networks and Wireshark Instructor

Website: www.thetechfirm.com

A Senior Network Specialist with experience in performance testing, network design, implementation, and troubleshooting LAN/WAN/Wireless networks, desktops and servers since 1989.

Tony has taught at Colleges/Universities, Networld/Interop and many onsite corporate settings to thousands of analysts.

Tony is an authorized and certified Fluke Networks and Wireshark Instructor. His Pine Mountain Group CNA Level I and II certification demonstrates his vendor neutral approach to network design, support and implementations.

Tony has architected, installed and supported various types of Residential Wireless High Speed as well as hundreds of WIFI hotspots. Tony combines custom programs, open source and commercial software to ensure a simple support infrastructure.

Tony works on networks from 2 to 120,000 nodes and specializes in post installation performance/design review. This process involves using various tools (Protocol analyzers, traffic generators and network management) and working on multi-vendor equipment (switches, routers, servers, etc).

Tony works at customer sites within a range of capacities from project management, network design, consulting, troubleshooting, designing customized courses and assisting with installing physical equipment.
Capturing Traffic

Capture Engine

Capture Filters

Winpcap – Airpcap - Libpcap

Network
Processing Packets

GiMP Toolkit (GTK)

Dissectors – Plugins – Display Filters

Core Engine

Capture Engine

Wiretap Library
Overview of HTTP

- Hypertext Transport Protocol
- RFC 2616 (HTTP v1.1)
- Distributed hypermedia information distribution application
WinInet is an API used for applications to use specific protocols like Gopher, FTP, and HTTP protocols to access Internet resources.

WinInet limits the number of simultaneous connections that it makes to a single HTTP server. If you exceed this limit, the requests block until one of the current connections has completed. This is by design and is in agreement with the HTTP specification and industry standards.

WinInet limits connections to a single HTTP 1.0 server to four simultaneous connections and connections to a single HTTP 1.1 server are limited to two simultaneous connections.

The HTTP 1.1 specification (RFC2616) mandates the two-connection limit. The four-connection limit for HTTP 1.0 is a self-imposed restriction that coincides with the standard that is used by a number of popular Web browsers.

You can configure WinInet to exceed this limit by creating and setting the following registry entries:

**Note** By changing these settings, you cause WinInet to go against the HTTP protocol specification recommendation. You should only do this if absolutely necessary and then you should avoid doing standard Web browsing while these settings are in effect:

- MaxConnectionsPerServer REG_DWORD (Default 2)
  Sets the number of simultaneous requests to a single HTTP 1.1 Server

- MaxConnectionsPer1_0Server REG_DWORD (Default 4)
  Sets the number of simultaneous requests to a single HTTP 1.0 Server
Since HTTP 1.1 uses a keep-Alive, this option is no longer required and ignored.

<table>
<thead>
<tr>
<th>Hypertext Transfer Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>GET / HTTP/1.1</td>
</tr>
<tr>
<td>Host: <a href="http://www.thetechfirm.com">www.thetechfirm.com</a></td>
</tr>
<tr>
<td>User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US)</td>
</tr>
<tr>
<td>Accept: text/xml,application/xml,application/xhtml+xml,application/xml;text/html;q=0.9,text/html;q=0.8,text/xml;q=0.7,text/plain,<em>/</em>;q=0.5,text/html;direction=rtl;q=0.8,text/html;direction=ltr;q=0.7,text/plain;direction=rtl;q=0.6;text/plain;direction=ltr;q=0.5,text/plain;direction=rtl;q=0.4;text/plain;direction=ltr;q=0.3, <em>/</em>;q=0.2</td>
</tr>
<tr>
<td>Accept-Language: en-us,en;q=0.5</td>
</tr>
<tr>
<td>Accept-Encoding: gzip, deflate</td>
</tr>
<tr>
<td>Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7</td>
</tr>
<tr>
<td>Keep-Alive: 300</td>
</tr>
<tr>
<td>Connection: keep-alive</td>
</tr>
<tr>
<td>Cookie: aqusr=A1044.380258BB021B29EB59</td>
</tr>
</tbody>
</table>
HTTP Capture Filter

Helpful when HTTP uses TCP Port 80

Helpful when HTTP is not using TCP Port 80
HTTP Display Filters

You can use the following Display Filters:

http *(TCP SYN, ACKs, RST or FIN packets will not be displayed)*

tcp.port==80 *(or whatever port number you use)*

http.request.method == "GET"

http.request.method == “POST"
List of HTTP Display Fields
HTTP Packet Structure

TCP-based (Port 80)
Variable-length header

[http-post.pcap]

```
GET /apps/Agent/en-us/Agent5/chknews.asp?affid=8
Request Method: GET
Request URL: /apps/Agent/en-us/Agent5/chknews
Request Version: HTTP/1.1
Accept: */*
User-Agent: MChip
Host: us.mcafee.com

GET to read data
```

```
POST /apps/agent/submgr/appinstru.asp HTTP/1.1
Request Method: POST
Request URL: /apps/agent/submgr/appinstru.asp
Request Version: HTTP/1.1
Accept: */*
ReportingSource: 1DCC47C7BB09444fBA3445D72BD98C0B
ReportingId: 1
Content-Type: application/octet
User-Agent: MChip
Host: us.mcafee.com
Content-Length: 526

POST to write data
```
Wireshark Field Names

As you highlight various fields in the Packet Details View, you’ll see the field name displayed in the status bar.
Analyze Normal HTTP Traffic

Most common commands:
- GET
- POST

Responses are numerical:
- 1xx  Informational
- 2xx  Successful
- 3xx  Redirection
- 4xx  Client error
- 5xx  Server error

Complete list of HTTP status Codes may be found at http://www.iana.org/assignments/http-status-codes or RFC2817
Dissecting HTTP GET *(http-post.pcap)*

- **Command GET, POST, etc..**
- **Uniform Resource Identifier may be classified as a locator (URL) or a name (URN) or both**
- **What media types are Accepted**
- **What Software made the call**

**Hypertext Transfer Protocol**

- **GET /apps/Agent/en-us/Agent5/**
- **Request Method:** GET
- **Request URL:** /apps/Agent
- **Request Version:** HTTP/1.1
- **Accept:** */*\n
**User-Agent:** MCUPDATE\n
**Host:** us.mcafee.com\n
\n
HTTP Connection

The ‘Connection;’ default is typically *keep-alive*, but a server may request the client to close the connection with a ‘*close*’ response.

```
GET /coop/images/google_custom_search_sinan.gif HTTP/1.1
Host: www.google.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv)
Accept: image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://www.thetechfirm.com
Cookie: _utma=173272373.2018293701.1186276722.11879197
```

```
HTTP/1.1 301 Moved Permanently
Date: Sun, 07 Jan 2007 08:23:36 GMT
Server: Apache
Location: http://espn.go.com
Content-Length: 227
Connection: close
Content-Type: text/html; charset=iso-8859-1
```
## Dissecting HTTP Response

### HTTP Version
- HTTP/0.9, 1.0, 1.1

### 10 Bytes of Data
- Content-Length: 10
- Content-Type: text/html
- Set-Cookie: ASPSESSIONIDASABTCQQC=0
- Cache-control: private

### Only receiving device can cache this data
## HTTP Commands, RFC’s and Status Codes Reference

### Commands:

<table>
<thead>
<tr>
<th>Method</th>
<th>References</th>
</tr>
</thead>
<tbody>
<tr>
<td>DELETE</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>GET</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>HEAD</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>LINK</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>RFC 2068</td>
</tr>
<tr>
<td>PATCH</td>
<td>RFC 2068</td>
</tr>
<tr>
<td>POST</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>PUT</td>
<td>RFC 1945</td>
</tr>
<tr>
<td>TRACE</td>
<td>RFC 2068</td>
</tr>
<tr>
<td>UNLINK</td>
<td>RFC 1945</td>
</tr>
</tbody>
</table>

### Status code categories:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1yz</td>
<td>Informational.</td>
</tr>
<tr>
<td>2yz</td>
<td>Success.</td>
</tr>
<tr>
<td>3yz</td>
<td>Redirection.</td>
</tr>
<tr>
<td>4yz</td>
<td>Client error.</td>
</tr>
<tr>
<td>5yz</td>
<td>Server error.</td>
</tr>
</tbody>
</table>
HTTP Filter Reference

• Capture Filter
tcp port 80

• Display Filter
Status line
Expressions
Documentation
HTTP Packet Counter Information

A good start to analyzing HTTP is to document the number of Get commands a webpage produces. From the Statistics HTTP menu choose Packet Counter. Print screen is the only way to capture this information.
Print screen is the only way to capture this information.
Analyze Unusual HTTP Traffic

http-espn.pcap

Notes:

Dependencies on other web sites.

Not all HTTP requests are successful.
HTTP Statistics

Load Distribution

Requests

Packet Counter
Round Trip Time Graphs

Very high round trip times!
TCP Stream Graphs

GET / HTTP/1.1
Accept: */*
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1; .NET CLR 1.0.3705; .NET CLR 1.1.4322; Media Center PC 4.0)
Cookie: user_id=4592139c-00042-01ba1-6289bce
RSP_DAEON=4dd70caa9c3c3e0eb6b6df82a86f8def9
Connection: Keep-Alive
Host: hp-laptop.aol.com

HTTP/1.0 200 OK
X-RSP: 1
Pragma: no-cache
Cache-Control: no-cache, no-store, private, max-age=0
Expires: 0
Set-Cookie: cobr=hp-laptop.aol.com; DOMAIN=.aol.com; PATH=/
Set-Cookie: POP_COOKIE=cbm=ab2RkMDExMTY3NTk2MTOeA%3d%
3d;Path=/;Domain=aps.web.aol.com;Expires=Tue, 30 Dec 2008 20:15:43 GMT
MIME-Version: 1.0
Date: Sun, 31 Dec 2006 20:15:43 GMT
Server: AOLserver/4.0.9b
Content-Type: text/html; charset=utf-8
Connection: keep-alive
Content-Encoding: gzip
Content-Length: 12133

client communications in red (by default)

server communications in blue (by default)
If your HTTP traffic uses a different port than TCP 80, use the Analyze-> Decode As feature.
Using the Round Trip Time Graph

unacceptable latency times
TCP Receiver Congestion

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>25</td>
<td>0.209709</td>
<td>61.8.0.17</td>
<td>10.0.52.164</td>
<td>HTTP</td>
<td>Continuation or non-HTTP traffic</td>
</tr>
<tr>
<td>26</td>
<td>0.210050</td>
<td>61.8.0.17</td>
<td>10.0.52.164</td>
<td>HTTP</td>
<td>Continuation or non-HTTP traffic</td>
</tr>
<tr>
<td>27</td>
<td>0.216069</td>
<td>10.0.52.164</td>
<td>61.8.0.17</td>
<td>TCP</td>
<td>2550 &gt; http [ACK] Seq=0 Ack=26280 win=730</td>
</tr>
<tr>
<td>28</td>
<td>0.241568</td>
<td>61.8.0.17</td>
<td>10.0.52.164</td>
<td>HTTP</td>
<td>Continuation or non-HTTP traffic</td>
</tr>
<tr>
<td>29</td>
<td>0.247779</td>
<td>61.8.0.17</td>
<td>10.0.52.164</td>
<td>HTTP</td>
<td>Continuation or non-HTTP traffic</td>
</tr>
</tbody>
</table>

Wireshark: Expert Infos

- Sequence: TCP Zero window: 7
  - Packet: 30, 32, 34

- Sequence: TCP Keep-Alive: 3
  - Packet: 30, 32, 34

- Sequence: TCP Keep-Alive: 6
  - Packet: 30, 32, 34

- Sequence: TCP Window Update: 2
  - Packet: 30, 32, 34

Bytes captured:

1609 (08:00:48:6f:4a:3a:09), Dst: 00:04:75:c9:51:b6 (00:04:75:c9:51:164 (10.0.52.164), Dst: 61.8.0.17 (61.8.0.17)

TCP Port: 2550 (2550), Dst Port: http (80), Seq: 0, Ack: 29200, L
This feature allows you to review all the files retrieved as well as rebuilding those files.
Other things to consider..

When troubleshooting, analyzing or baselining HTTP, you should monitor the following additional protocols.

- DNS
- WINS
- LDAP
- Proxy server communication

Noting if data submitted (POSTED) to server is in clear-text or not, is very helpful.