1. Wireshark



Wireshark, doo doo doo doo doo doo.

1.1. What is it?

Wireshark is an open-source packet analysis suite. I use it for network troubleshooting, and debugging software that uses a network to communicate.

You can even use it to monitor traffic over a bluetooth interface, or USB.

I call it a suite for the sole reason that it comes with a bunch of utilities.

1.1.1. Anecdote

In the summer of 2001, I had the opportunity to take an SGI Indy home from high school. Nobody had the password.

The machine rain IRIX, which was popular at the time for servers, I was convinced that I could gain access to the machine if I could find it via the IP address.

There were two problems:

* The Monitor was too big to bring home, and the keyboard was spotty.
* It was nice enough to have a standard ethernet port.

What I tried:

* My first attempt was plugging it in to the router at home which should assign an address to it via DHCP. The original purpose of this machine was a gateway to the internet for the highschool, chances are this would not work.
* Tried plugging it into a switch with my DHCPD running to see if it would assign IP addresses to machines, it did not.

The next thing I tried, was **ethereal**, something that I read about on slashdot and figured this was what what needed.

If we rewind back to the 2000s, this was wiresharks original name.

With the cable connected, I started wireshark from my XFCE terminal, and pointed it at the /dev/eth1 device. The AMD Athlon 350mhz computer with a 100mhz frontside bus was quick for it's time, but this still took up a lot of CPU and disk.

A bunch of traffic started coming through. This was the first time that I had ever used this program and I was learning what everything meant.

A brief time later, I stopped the trace, I had what I needed. There was a request that said:

Who has 192.168.0.3 tell 192.168.0.1

After settting my IP Address to 192.168.0.3, then subseqently sending a ping request to 192.168.0.1, I got a response.

Opening a telnet connect to 192.168.0.1, I received the familiar IRIX telnet logon session.

At this point I took the version number it was displaying and slightly changed a really popular telnet exploit to work against this system, and then I had root access.

(<https://www.securityfocus.com/bid/1572/info>)

Using the man pages to figure out how to add users to the system, I then added a backup acccount, and made a setuid program so that I could login locally later.

The rest of the summer I spent playing trying to crack the password, and then playing with IRIX and trying to get the remote X-Window session working. It was fun, and I could not have done it as easily without Ethereal (wireshark).

The backup plan was to connect the SCSI hard drive to my computer and use a HEX EDITOR to change the root password to a known hash, allowing me to login without a password. This is the method on a SOLARIS server I bought a many years later from a different school board. Efficient.

1.2. tshark

This is the command line version of wireshark and allows you to capture things "Headless".

1.2.1. Honorable Mention: netsh trace start

In Windows 2012 R2+, you can capture traffic without addition utilities installed by using the **netsh trace** command.

@echo "Starting Trace:"

netsh trace start persistent=no capture=yes tracefile=c:\temp\nettrace.etl"

@echo "Waiting 30 seconds"

timeout 30

@echo "Stopping Trace"

netsh trace stop

*To use this file in wireshark, you will have to convert it. Thanks Benjamin Perkins:*[*https://blogs.msdn.microsoft.com/benjaminperkins/2018/03/09/analyze-netsh-traces-with-wireshark-or-network-monitor/*](https://blogs.msdn.microsoft.com/benjaminperkins/2018/03/09/analyze-netsh-traces-with-wireshark-or-network-monitor/)

1.3. libpcap

The Library for Packet Capture is shared by a few open source projects, most people will be familiar with tcpdump or ngrep.

1.4. What does it do?

Using libpcap it can hook into network interfaces and capture traffic. Depending on your operating system, and what kind of network card(s) you have, you might be able to do advanced stuff.

You will generally be able to do the basics like see local traffic.

1.5. Network Analysis

1.6. Search individual frames for a particular string

frame contains ca.yahoo.com

1.7. How do I see HTTPS Traffic?

On Windows, using either Firefox or Chrome, you'll be able to set the following environmental variable:

SET SSLKEYLOGFILE=%USERPROFILE%\sslkeysENV.pms

1.7.1. Who is on the network?

If you are curious who is broadcasting on a network, you can filter by

arp

This will display any machines that are broadcasting with the ARP protocol.

1.7.2. Filter SSL and HTTP\* traffic

This filter will look for all SSL traffic, and find anything that is HTTP2 or HTTP.

(ssl) && ((http2) || (http))

When you find a stream you are looking for, you will see 1-4 columns at the bottom of the Packet Bytes window:

* Frame
  + The original data frame.
* Decrypted TLS
  + The Decrypted TLS stream
* Reassembled body
  + The reassembled body (all packets in this request)
* Uncompressed entity body
  + If the content is compressed (it most likely was), this step will decompress it making it human readable.

Because of the the complexty and limited nature, for HTTP I much prefer Fiddler for analyizing HTTP/HTTPS traffic.

1.8. MATE

The Meta Analysis and Tracing Engine (MATE) is my goto extension to configure on a new Wireshark install.

*Examples taken from:*[*https://wiki.wireshark.org/Mate/Examples*](https://wiki.wireshark.org/Mate/Examples)

*Cloudshark Link:*[*http://cloudshark.org/captures/9279c75f8161*](http://cloudshark.org/captures/9279c75f8161)

MATE groups everything in Protocol Data Units (PDUs), a PDU is a specific block of information transferred over a network. [*https://techterms.com/definition/pdu*](https://techterms.com/definition/pdu)

1.8.1. Sessions that last less longer than one second

mate.tcp\_ses.Time > 1

This is great for narrowing down any type of sessions which don't last long. If you are looking for timeouts or other types of issues, this will be valuable.

1.8.2. Tcp sessions that have less than 5 packets.

mate.tcp\_ses.NumOfPdus < 5

A TCP session has a triple handshake, and usually at least a few exchnages after that. So if we search for sessions with less than 5 Packets, chances are it's an unreachable connection or something that's blocked off.

1.8.3. Packets for the third tcp session MATE has found:

*This is the (Stream Index - 1)*

mate.tcp\_ses == 1

1.8.4. Show all unsuccessful TCP connection attempt retries.

From the MATE documentation:

[*https://osqa-ask.wireshark.org/questions/10640/how-to-find-syn-not-followed-by-a-synack/10758*](https://osqa-ask.wireshark.org/questions/10640/how-to-find-syn-not-followed-by-a-synack/10758)

(mate.tcp\_ses.NumOfPdus < 4) && (tcp.flags eq 0x02 && tcp.analysis.retransmission)

This filter adds to the previous ones, we look for any retries, and specifically packets that are less than 4. If we make this greater than 4, we can actually find packet loss in a particular connection.

1.9. Chaining the Filters

To make the most use out of the filters, chaining them becomes nessessary.

1.9.1. Group with Parenthesis

**Yes**

(tcp) || (dns)

**No**

tcp || dns

1.9.2. Negate outside of Parenthesis

**Yes**

!(tcp)

**No**

(!tcp)

2. Fiddler

Fiddler is my goto software for HTTP Request debugging on Windows. It's always open on my computer for the off chance that I view a site that does something really iffy, and want to know more.

2.1. Easy Mode for HTTP centric requests

On the Windows Eco-system it works flawlessly and allows you to intercept traffic on the top 3 browsers.

2.2. Alternatives

There are a lot of programs that do things that Fiddler does, but in my humble opinion, Fiddler is king for people of all skill levels.

2.2.1. burp suite

Burp is awesome, but I don't really want to have this installed on random servers, and gosh the IT people get in a kerfuffle when they see this. <https://portswigger.net/burp>

2.2.2. mitmproxy

The name alone is scary, it's pretty much a tuned down version of fiddler from a GUI perspective, but incredibly powerful. <https://mitmproxy.org/>

3. Fiddler & HTTP

Out of the box, Once fiddler starts up it should start intercepting traffic. You might have to restart your browers or web applications to see this information.

3.1. NET Applications

In order to receive information about web requests sent to or from a .NET Application, you can try modifying the machine.conf, the web.config, or the application.config

3.2. Java Applications

I'm actually pretty java dumb, but this command usually works:

java -xproxy=

3.3. Fiddler & HTTPS

By default, HTTPS decryption is not enabled, for a very good reason. To enable it you must accept a root certificate that is generated by Fiddler. this root certificate allows the traffic that is intercepted from between your browser and the remote servers to be signed by something that the computer recognizes.

3.4. Capture and Decrypt HTTPS Traffic

In Fiddler

* Open the Tools Menu, and select "Options"
  + Click on the "HTTPS" tab
    - Select "Capture HTTPS CONNECTs"
    - Select "Decrypt HTTPS traffic"
      * Click yes to every message. There may be a lot of them.

This process is much less painful than Wireshark. Since Fiddler is centered around the HTTP protocol, you get a lot more information as well.

3.5. Page Load Metrics

Visiting the Wookieepedia entry for R2-D2 (<https://starwars.fandom.com/wiki/R2-D2>) we can click on the request

Request Count: 1

Bytes Sent: 1,006 (headers:1,006; body:0)

Bytes Received: 114,097 (headers:1,039; body:113,058)

ACTUAL PERFORMANCE

--------------

ClientConnected: 21:55:13.453

ClientBeginRequest: 21:55:13.914

GotRequestHeaders: 21:55:13.914

ClientDoneRequest: 21:55:13.916

Determine Gateway: 0ms

DNS Lookup: 0ms

TCP/IP Connect: 0ms

HTTPS Handshake: 0ms

ServerConnected: 21:55:13.507

FiddlerBeginRequest: 21:55:13.917

ServerGotRequest: 21:55:13.917

*These two metrics tell us when the Client Sent the Request, and the Server Received the Request. They are the same value, which is good performance wise.*

ServerBeginResponse: 21:55:13.962

*This is when the server starts transferring data. If there is a large delay here, it's usually a few things:*

* *The Web Server*
* *A Backend Server (Database, File Server, etc)*
* *A network problem (Open up Wireshark, check for packet issues.)*

GotResponseHeaders: 21:55:13.962

ServerDoneResponse: 21:55:14.145

ClientBeginResponse: 21:55:13.962

ClientDoneResponse: 21:55:14.145

*The time between ServerGotRequest and ServerBegingResponse is how long the request took to process*

*The time between ServerBeginResponse and ClientDoneResponse can be considerered the transfer time*

Overall Elapsed: 0:00:00.230

*That's 0.23 seconds for this individual request*

RESPONSE BYTES (by Content-Type)

--------------

text/html: 113,058

~headers~: 1,039

*The total size of the body (text/html) and the total size of the headers.*

3.5.1. TTFB & TTLB

You'll notice that I have these two columns added by default in my view, as a performance person, I'm really interested in these values.

* TTFB is Time to First Byte
  + The time it takes for the server to start sending data. I like this number to be as low as possible.
* TTLB is Time to Last Byte
  + The time it takes for the server to send it's last byte of data. You subtract TTFB from TTLB, to get the Total Transfer Time.

You can unreliably extrapolate a transfer speed from these two values. This helps remove any network speed problems from determinig why an application is slow.

**3.5.1.1. (TTLB - TTFB) = Transfer Speed**

**3.5.1.1.1. Scenario 1 - High TTLB, Low TTFB**

If TTLB is 5000 and TTFB is 1000 and the total bytes sent is 15,360 (15 megabytes) We can assume that it took

* 1 second to process the data.
* 4 seconds to transfer the data.

We can safely assume that the network connection is just slow. Run the request locally on the web server if possible, this removes any intermediary things like proxies or firewalls.

**3.5.1.1.2. Scenario 2 - High TTLB, High TTFB**

If TTLB is 5000 and TTFB is 4990 and the total bytes sent is 15,360 (15 megabytes) We can assume that it took

* 4.99 seconds to process the data.
* 0.01 seconds to transfer the data.

The network is probably not to blame, do an end to end inspection just in case.

3.6. Manual Replay

Fiddler allows manual replays of Web Requests, which I feel is one of it's lesser used features.

3.7. Auto Replay

One of my favorite parts of Fiddler, is the tamper and replay. I'll cover a very simple use case for this with twitter.

My favorite scenario was when at a previous job working for one of the provinces, we had a contractor create a client server application. They didn't put any user sanity checks into the REST apis. This meant when the application sent a permissions request, and the server sent back "User", if I changed that to "Admin", I could see all of the Administrative functions.

Although this seems like something incredibly simple, how do you test for it without writing code? Using any of the available http tamper proxies, or my favorite, fiddler.

3.7.1. Steps to Reproduce

* Step 1: Capture and Edit the Users Recommendation request in Twitter (<https://twitter.com/i/users/recommendations>)
* Step 2: Visit the likes page (<https://twitter.com/i/likes>)

Expected Result: I'll see 98 as the notification count.