

Packet School 101 - Part 1

Over the course of the next few weeks I am going to be putting out a series on network traffic analysis. This is one of the most important skills any network administrator can have and is absolutely crucial to solving a variety of network related problems.

We won't get into any actual packet analysis in the introductory article We will, however, go ahead and make sure we have the appropriate software to proceed, along with a brief understanding of how it works and how to use it.

Getting Equipped

The software we will be using for this series is the ever popular Ethereal network sniffing application. This program is distributed freely and can be downloaded at <http://www.ethereal.com>. Installing this software is simple enough that I am not going to go through it here, although it is important to note that this software relies on the WinPcap driver which is included in the installation package. As a lot of you may already know, there have been some issues with WinPcap and version skew, as in different versions of the driver being required for different pieces of software to work with it. This being the case, I recommend you to install the version of WinPcap that comes with Ethereal even if you already

have a version of this driver installed. This will help to prevent any possible issues in the future.

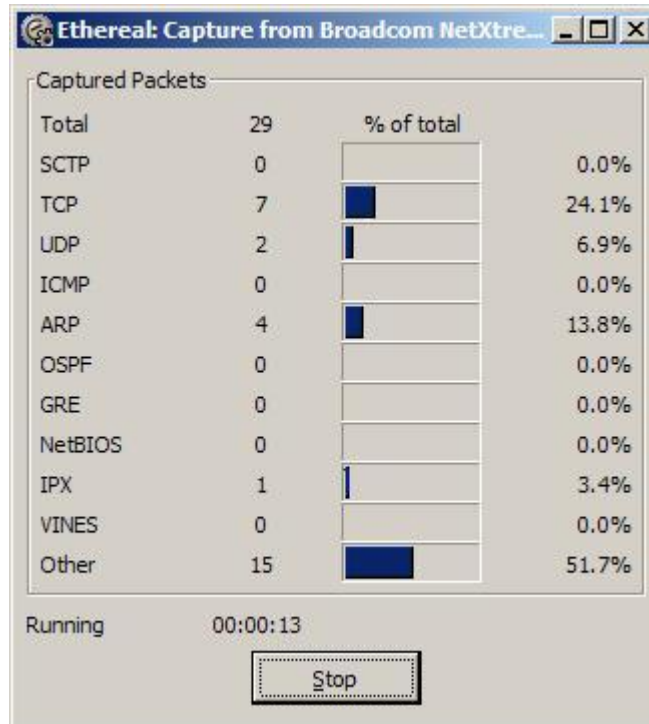
There is no actual hardware required for doing this type of analysis, however, it makes things a LOT easier if you have an old 10/100 hub lying around. I will explain why this is here very shortly.

Ethereal Basics

Once you have installed Ethereal along with the WinPcap driver you should be ready to dive in head first. The first thing we are going to do is a simple packet capture. To do this you will first need to select your capture interface by clicking the "List available capture interfaces" button to the far left hand side of the main toolbar.



Once you have done this you will be presented with a window listing all available capture interfaces. Clicking the "Capture" button next to the interface you wish to use will begin capturing packets from this connection.



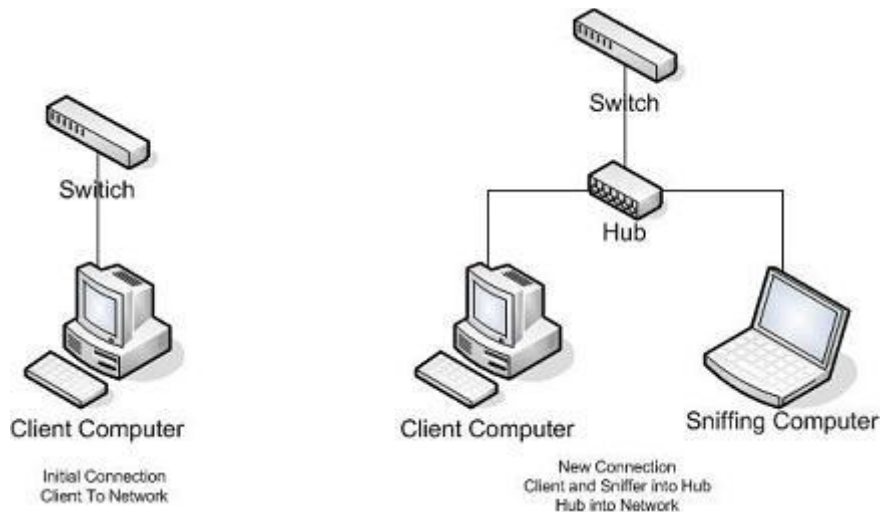
Wait for a few minutes until a significant amount of packets have been collected and then click the "Stop" button. You should be returned back to the main screen with a whole bunch of new data. Congratulations! You have just completed your first successful packet capture!

Slipping On To the Wire

Now that you know how to do a basic packet capture in Ethereal it is important to learn how to capture the right traffic. Assuming you are on a switched ethernet network (which most everybody is these days), all of the traffic you just captured was your own. That is, all traffic was either coming to or going from the computer from which you initiated the packet capture. This is basically how a switched

network functions. The switch only sends data to those ports in which it is destined. This brings up the question of how do you capture traffic from a computer that packet sniffing software is not installed on?

Remember earlier when I told you that it would come in handy to have an old 10/100 hub lying around? Well this is that time. I am not going to go into great detail on the differences between a switched and a hub network, but put simply, in a hub network every piece of data is sent to every single port. The hub uses a lower level of logic in that it does not know where data is supposed to go, so it sends it to every client on every port and lets those clients accept it if they want it and deny it if they don't. This means on a hub network when you sniff packets you are seeing the packets of the entire network. So lets say you are trying to troubleshoot a computer on a switched network. What you will want to do is go to the client computer and unplug it from the network, and plug it into your hub, along with the computer you will be using to sniff the traffic. From here you will plug the hub onto the network using its uplink port. This will effectively put the two computers you will be using onto a hub network allowing you to read all the traffic you need to. See the diagram below for a visual explanation of the hub setup.



Homework

In the next section of this series we will look at some basic packet structure as well as some more features of Ethereal. We will also look into some basic troubleshooting using packet analysis. Between now and then I would highly recommend playing around with Ethereal and looking at your own networks packets to see if you can try and figure out what some of them are.

[Packet School 101 - Part 2](#)

7/5/06 - If you like this series, [digg it!](#)

****UPDATE****

Enjoy Packets

Packet School 101 - Part 2

If you are reading this then it means you

probably enjoyed the first part of my series on packet analysis, so if that is the case, welcome back! In this installment we are going to analyze our first trace file to learn some more packet analysis basics. I am going to assume that you have played around with Ethereal a bit and am going to forgo the explanation of all the different windows within it and things of that nature. Before we start you are going to need to download the sample trace file we will be working from. I obtained this trace file as well as a lot of others I will be using from Laura Chappell, Sr. Protocol Analyst for the Packet Level Protocol Analysis Institute (<http://www.packet-level.com>).

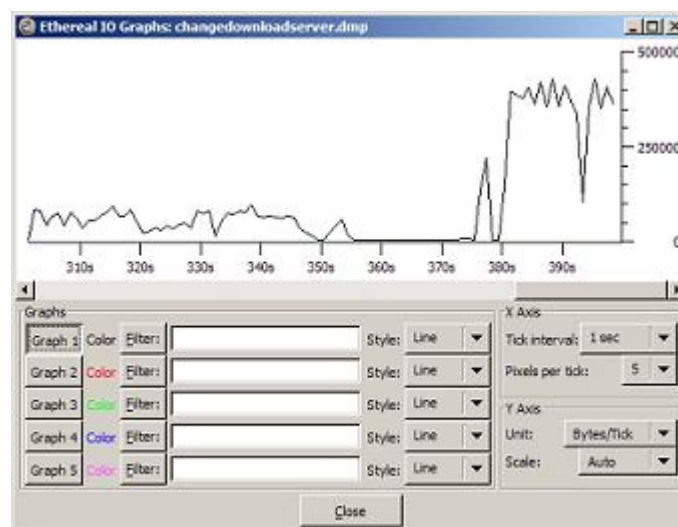
Our First Analysis - Downloading a File

[Click here to download sample trace file \(approx. 13MB\)](#)

Downloading a file is a pretty basic function when described at the network/transport layer. A client request a file from a server via HTTP and the file responds with this file. This takes the form of several packets as the file downloaded is segmented into various size packets for transport across the networks it must traverse to reach its destination. In our trace file we are going to look at an example of downloading a file from two different servers. The first server we will look at is going to be an extremely slow server so that we can look at some of the various problems that Ethereal will

show us. After this we will look towards the end of the trace file for an example of what a healthy file transfer looks like.

The first thing we want to look at is the I/O graph of the packet capture. The I/O graph is customizable with a lot of various options and is a great way to visual interpret some of the data you have captured. To view the I/O graph of this packets capture click "Statistics" on the menu bar and then select "IO Graphs". Once you have done this you are going to need to look towards the bottom right corner of the screen and change the "packets/tick" option to "bytes/tick". This will provide the information to us in a form that is much more useful. You may feel free to change any other visual settings in this window.



Looking at the I/O graph for our packet capture we notice that traffic is slightly elevated near the beginning of the capturing before flattening out. Then towards the end of the capture traffic

spikes to a new high with a slight drop in the middle. Knowing that we are looking at an example of a fast and slow download we can easily distinguish where the two downloads take place just by looking at this graph. Let's look at the packet capture and see what else we can determine about these downloads.

Scrolling through all of the packets you will see a lot of various information. The biggest portion of this is normal traffic and is therefore useless to us as we are only looking for abnormal traffic. In order to filter this abnormal traffic out we are going to use a new feature in Ethereal called the expert info window. To open this window you can simply click on "Analyze" in the menu bar and select "Expert Info". By default, the expert info window is going to show us all warning, error, note, and chat communication. Chat communication does not really constitute abnormal traffic, therefore we are going to change this settings to show only warnings, errors, and notes by changing the appropriate setting under the "Severity filter" heading.

No.	Event	Queue	Protocol	Summary
27	Note	Sequence	TCP	Window update
32	Note	Sequence	TCP	Window update
63	Note	Sequence	TCP	Window update
120	Note	Sequence	TCP	Window update
162	Note	Sequence	TCP	Window update
165	Note	Sequence	TCP	Window update
185	Note	Sequence	TCP	Window update
219	Note	Sequence	TCP	Window update
330	Note	Sequence	TCP	Window update
399	Note	Sequence	TCP	Window update
220	Note	Sequence	TCP	Window update
265	Note	Sequence	TCP	Window update
270	Note	Sequence	TCP	Window update
274	Note	Sequence	TCP	Window update
282	Warn	Sequence	TCP	ACKed last segment (common at capture start)
334	Note	Sequence	TCP	Window update
338	Note	Sequence	TCP	Window update
346	Note	Sequence	TCP	Window update
350	Note	Sequence	TCP	Window update
374	Note	Sequence	TCP	Window update
380	Note	Sequence	TCP	Window update
387	Note	Sequence	TCP	Window update
398	Warn	Sequence	TCP	Previous segment lost (common at capture start)
399	Note	Sequence	TCP	Duplicate ACK (#1) to ACK in packet #397
400	Warn	Sequence	TCP	Retransmission (suspected)
402	Note	Sequence	TCP	Window update
409	Warn	Sequence	TCP	Previous segment lost (common at capture start)
410	Note	Sequence	TCP	Duplicate ACK (#1) to ACK in packet #400
411	Warn	Sequence	TCP	Retransmission (suspected)
413	Note	Sequence	TCP	Window update
432	Warn	Sequence	TCP	Previous segment lost (common at capture start)
433	Note	Sequence	TCP	Duplicate ACK (#1) to ACK in packet #431
434	Warn	Sequence	TCP	Retransmission (suspected)
436	Note	Sequence	TCP	Window update

Anatomy of a Slow Download

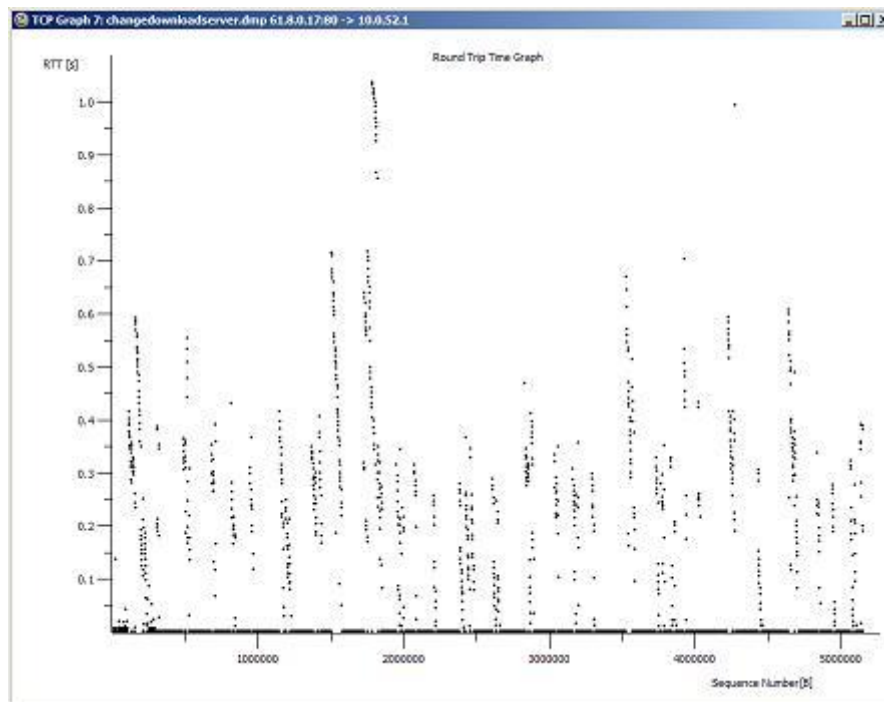
Looking at the expert information window there are a few things we want to discuss right away. The first thing you will see in abundance are TCP Window Update packets. If you are familiar with TCP/IP you know that the transmission rate of data is determined by the size of the TCP receive window. When transferring data between clients they will constantly send TCP Window Update packets to each other as their ability to receive data speeds up or slows down.

The next thing we want to look at is where we have our first problematic packets. As the download starts you begin to see "Previous segment lost" packets. Basically what this is

saying is that during the course of data transfer, all of a sudden a packet is dropped. In response to this the client sends what you are seeing as "Duplicate ACK" packets to the server in order to request to packet that was dropped. The client will continue sending these duplicate ACKs until it finally receives the packet it was looking for. The retransmission of this missing packet is what you are seeing as "Fast retransmission" in the expert info window. At the start of the download you are only seeing one or two duplicate ACKs at once but as the download progresses you begin seeing more and more of them which means you are experiencing more latency. At this point you can browse through the rest of the capture and see that it is riddled with these segment losses and duplicate ACKs. If you browse to the end of the capture file however, you will notice this is not the case. In this instance there are only one or two dropped packets in the successful download.

One last thing we are going to look at is another way to visually represent the packet capture data called the "TCP Stream Graph". You can access this graph by clicking on a packet related to the stream you wish to analyze (in the case you can select packet number 1023) and going to "Statistics", selecting "TCP Stream Graph" and clicking on "Round Trip Time Graph". This graph is not too visually pleasing but is a great way to compare

round trip time (RTT) throughout a packet capture. You will notice near the beginning of the capture with the slow download we are seeing RTT of up to one full second. This is complete unacceptable for downloading a file. Even when downloading a file off of the internet you should see times at no more than 0.1 seconds. An ideal RTT for data transfer across the internet is no more that 0.04 seconds (40 milliseconds). Looking at the beginning of the graph as compared to the high speed download at the end of it can produce some draw jopping results.



Homework

I would encourage you at this point to try taking some packet captures of downloading various files on the internet. See if you can find

a high utilization server somewhere so you can view the packets of a slow download you did yourself. This will help you to better understand what we learned today. In the next installment of Packet School 101 we are going to examine a trace file from a network in which there are router performance problems.

[Packet School 101 - Part 3](#)

[Packet School 101 - Part 3](#)

The response from this series has been tremendous! As of yesterday I have managed to make the front page of digg.com and make #2 on the list of the most linked sites on del.icio.us. I have seen my bandwidth grow exponentially and have logged over 5 GB of traffic in the past 24 hours. I have also had a lot of great comments regarding the first two parts that I hope to address later on in the series. I plan on ending with a Q & A so if you have any major questions feel free to e-mail me.

Troubleshooting a Slow Router

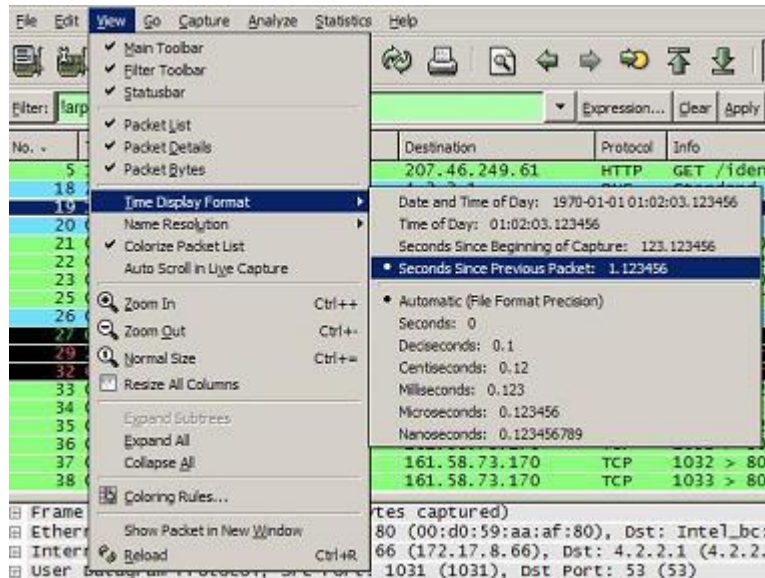
[Download the sample trace file by clicking here \(<1 MB\)](#)

In this section we are going to look at a client who is trying to connect to a website but it is experiencing all kinds of network slowness issues. Opening the sample trace file, the first thing you will notice is a lot of ARP broadcast packets. These are typical in a lot of network environments for layer 3 to layer 2 address resolution. For the purpose of what we are doing here, we are going to remove these from our trace file so that they don't clutter things up. You can do this by typing "!arp" in the filter text box near the top of the Ethereal window.

Getting on Time

At this point in our learning it is pertinent to take notice of the "Time" column in the main Ethereal window. You will see a time listed next to each packet, and by default, this shows the time the packet was received in relation to the beginning of the packet capture. This type of view has its purposes in some settings, however, for troubleshooting a slow network you will want to change this setting to display the time relative to the packet received previously. You can do this through by going to View > Time Display Format, and selecting "Seconds Since Previous Packet". In this new time view you will notice that the times are now displayed as the amount of time since the previous packet was captured. For example, packet 4 was received 1.728522 seconds after packet 3. This will be much more handy to us in our troubleshooting of the slow network

communication.



Looking for the Source of the Latency

Now that we have our time column set to display data to us in a much more helpful way, scrolling down through the trace file we see our first major network activity at packet 18 where the client (172.17.8.66) makes an HTTP request to get the website www.packet-level.com. Typically, the next packet we see should be the dns servers response to the client. In this case however, the server does not respond back to the client, so the next packet we see is over a second later and is the client attempting to request the webpage a second time. After this second request we finally see a response from the DNS server pointing us to the IP address of the web server in packet 20. In an ideal situation, as soon as this dns query is completed the client and

server should begin a standard TCP/IP handshake (SYN, SYN/ACK, ACK), and in our case the client does its part sending its initial SYN packet out within 4 milliseconds (third place from the decimal point is milliseconds) of receiving the DNS response. The server on the other hand responds an incredible amount slower taking over half a second to send its SYN/ACK reply. At this point we can definitely begin to see that it is something other than the client causing the network latency.

Interestingly enough as we continue down into the trace, in packet number 26 we see a second DNS reply from the server. This is the reply to our second DNS request that we made initially. The only problem is that it is about 5 seconds too late! Given that our client computer has already established a connection with the server there is no real need for this second connection, and it throws up an ICMP destination unreachable packet immediately following the receipt of the DNS response.

Going back to our already established connection to the server, we begin to see problems sprouting up. After making our initial TCP/IP handshake the client requests the actual content of the webpage at packet 25. Quite some time goes by and then in packets 29 and 32 we see two TCP Retransmission packets. In this case, the client has requested the webpage, not gotten a response, waited a certain amount of time, and sent a

retransmission to the server in order to make another attempt at getting the data. After the third retransmission we finally see a response from the server in packet 33. Now if you add up the times from packets 25 to 33 you will see it has taken us nearly 9 seconds to get the first bit of data from the webpage we are requesting. It doesn't take a packet analyzing expert to realize this is entirely unacceptable.

Fixing the Problem

Given the information we have just seen we know that the client is not at fault for the slow communication. The principal rule of thought for figuring out the problem location is to move upstream along the network. In this network, the next step would be to look at the router to see if it is malfunctioning in any way. Upon rebooting the router on this particular network, the speed of data communication increased tremendously and the problem was solved. However, if the problem had not been the router then you would need to move upstream to the router of the network in which the web server you are connecting to is behind. Unfortunately when that is the case you typically do not have the power over the remote network to do anything about it.

Homework

In our next installment we are going to look at a spyware infection and its effects on a

workstation. Now that you understand the concept of retransmissions and latency you may be asking yourself what exactly is a good time for something such as a website request to take place? The best thing to do in this case is to sniff the packets on your own network whenever you are not having any issues. Ideal communication times are going to vary for each and every network you are on so this is another case in which you will want to sniff your own network. Getting to know what the packets look like in your network when it is healthy will SURELY pay off in the future.

[Packet School 101 - Part 4](#)

In this segment of Packet School 101 we are going to take a look at the trace files of a computer infected with spyware and one suffering from an application fault.

Spyware Infection

In our first sample scenario, a user has called complaining that her computer is dropping its network connection three minutes or so after booting up. When this happens the computer also maxes out its processor utilization. After making some calls you verify that everybody else is running as they should be, so the problem is just isolated to this one computer.

When looking at the computer, it is apparent that her problem does exist as told. So what do we do? We fire up our sniffer of course!

Examining the Packets

[Click here to download the sample trace file \(< 1MB\)](#)

In examining our trace file there are a lot of problems with many things going on but we are only going to focus on the one client machine we are looking out down. It's IP address is 172.16.1.10. The first thing we notice at the beginning of the capture is that an outside IP address keeps trying to make a connection to our computer on port 26452. Our client doesn't know what to do with this traffic since it is not expecting it, and therefore sends repetitive RST packets effectively ending the communication. This obviously should not be happening, however it continues to do so until we get to packet 70 and see the client requesting a copy of a file called analiz.exe from this rogue IP address via TFTP. This file begins to download, and ironically enough, that is when the computer starts experiencing problems

```
TFTP Read Request, File: analiz.exe, Transfer type: octet
TFTP Read Request, File: analiz.exe, Transfer type: octet
TFTP Data Packet, Block: 1
TFTP Acknowledgement, Block: 1
TFTP Data Packet, Block: 1
```

The Resulting Analysis

After spending some time actually on the physical computer the process analiz.exe was

found to be running. After terminating this process and removing the file via an automated spyware scanning utility the computer began to run normally. What was happening in this case was that after a computer would boot up it would try to make a connection to the rogue IP address associated with this spyware. After a certain amount of time the client would then download an updated copy of this spyware file via TFTP and run it on the system.

This brings up several good security points to remember. The first one of these is the most obvious and that is to educate your users about spyware and how to avoid it. However, another key point to mention is that this infection did a lot of its dirty work through TFTP which is a UDP based protocol. A lot of the time when configuring firewall filtering rules most administrators will completely ignore UDP and only focus on TCP. You can not neglect UDP protocols when doing this, as a lot of spyware will use this weakness to completely hammer your network.

Application Fault

The next file we are going to examine is the trace of an application fault that is causing a client computer to run incredibly slow. In this scenario we have a client who is complaining that their application is not working because the network is running too slow.

Digging Deeper

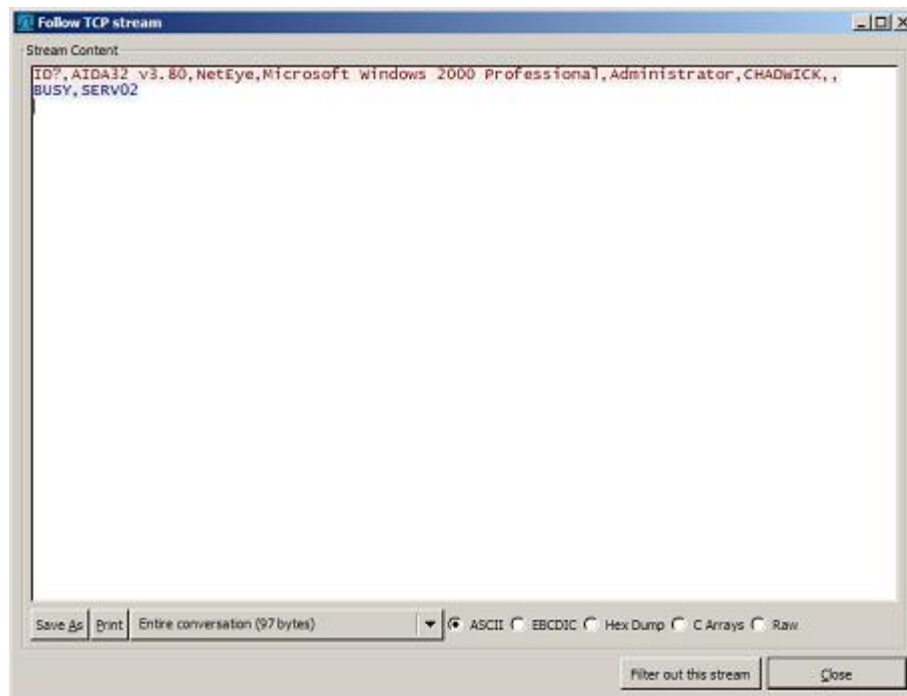
[Click here to download the trace file \(< 1MB\)](#)

When we open our trace file we can determine several things by looking at the first few packets. The first thing we see is communication taking place between our client and server machines. Looking at the packet times we can see those requests are happening at rates of less than a millisecond so that indicates that there shouldn't really be any latency on the wire itself. Also, we see that the client and server are sending their data back at more than adequate rates so there shouldn't be an issue with either of them acting sluggish.

When we get to packet 5 however, we begin to see something interesting. Our client machine is making a NetBIOS request to our server, and then in the following packet, our server (which is NOT running NetBIOS) returns an ICMP Destination Port Unreachable packet. So what is causing this NetBIOS traffic from our client? Well Wireshark provides a great interface for viewing data contained within a stream of data. In order to utilize this you will need to right click on packet number 4 and click "Follow TCP Stream". This will show us the data contained in that stream of communication, including upper layer protocol information.

When we look at this stream of data it is clear to us what is going on. The user is attempting

to use the AIDA32 application. Not only do we find out this information, but we can also see that they are doing this under the "Chadwick" user account which is in the administrators group of the local machine.



The Verdict

After looking at this TCP Stream data it is clear to see that the network is not at fault. What is happening is that the application is making a NetBIOS request to the server, and since the server isn't running NetBIOS it responds to the applications request by saying it is busy. In the design of the application which is being used, there is no formal mechanism in place that lets the user know that the server is not accepting its communication. This means that the application is just sitting there doing nothing

waiting for valid response, which it will never get. This being said, the proposed solution to this type of problem would be to enable NetBIOS on the server which the program is connecting to or find a program that uses an alternate means of communication.

Homework

Today we have analyzed to more trace files that have given us a further understanding of more packet related troubleshooting techniques. If you manage any type of network I would be willing to bet that you have problems with spyware on occasion. This being the case, the next time you see a computer with a spyware problem, sniff it's communication. You would be amazed to see how often various spyware applications will silently "phone home" to download updates to the infection and perform various other tasks. In the next installment of Packet School 101 we will take a look at what a trace file looks like when a computer is the target of a port scan and attempted denial of service attack.

****UPDATE****

[Packet School 201 - Part 1 \(ARP\)](#)

The ARP protocol was designed out of necessity to facilitate the translation of addresses between the second and third layers of the OSI model. The second layer, or data-link layer, uses MAC addresses so that hardware devices can communicate to each other directly on a small scale. The third layer, or network layer, uses IP addresses (most commonly) to create large scalable networks that can communicate across the globe. The data link layer deals directly with devices connected together whereas the network layer deals with devices that are directly connected AND indirectly connected. Each layer has its own addressing scheme, and they must work together in order to make network communications happen. For this very reason, ARP was created with RFC 826, "An Ethernet Address Resolution Protocol".

How it Works

The basic idea behind ARP is for a machine to broadcast its IP address and MAC address to all of the clients in its broadcast domain in order to find out the IP address associated with a particular MAC address. Basically put, it looks like this:

Computer A – "Hey everybody, my IP address is XX.XX.XX.XX, and my MAC address is XX:XX:XX:XX:XX:XX. I need to send something to whoever has the IP address XX.XX.XX.XX, but I don't know what their hardware address is. Will whoever has this IP address please

respond back with their MAC address?

All of the other computers that receive the broadcast will simply ignore it, however, the one who does have the requested IP address will send its MAC address to Computer A. With this information in hand, the exchange of data can begin.

Computer B – “Hey Computer A. I am who you are looking for with the IP address of XX.XX.XX.XX. My MAC address is XX:XX:XX:XX:XX:XX.

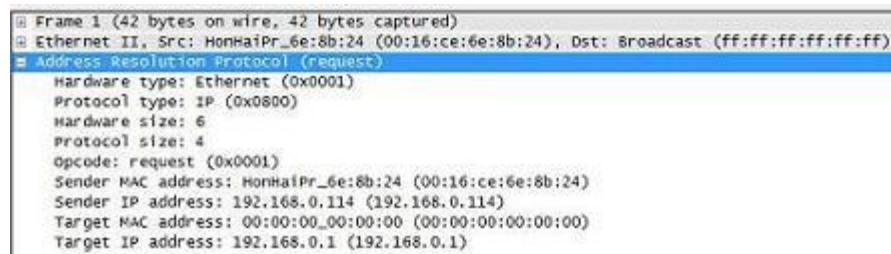
One of the best ways I’ve seen this concept described is through the limousine driver analogy. If you have ever flown, then chances are when you get off of a plane, you have seen a limo driver standing with a sign bearing someone’s last name. Here, the driver knows the name of the person he is picking up, but doesn’t know what they look like. The driver holds up the sign so that everyone can see it. All of the people getting off of the plane see the sign, and if it isn’t them, they simply ignore it. The person whose name is on the card however, sees it, approaches the driver, and identifies himself.

The Packet Level

Understanding the basic concept of ARP, we can take a look at some packets to see how it actually functions. Here we will step through

the entire ARP process, start to finish. In this scenario Computer A needs to communicate with Computer B. you can download this sample capture file [here](#).

Step 1: Computer A Generates Broadcasts an ARP Request Packet



```
Frame 1 (42 bytes on wire, 42 bytes captured)
Ethernet II, Src: HonHaiPr_6e:8b:24 (00:16:ce:6e:8b:24), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
Address Resolution Protocol (request)
  Hardware type: Ethernet (0x0001)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: request (0x0001)
  Sender MAC address: HonHaiPr_6e:8b:24 (00:16:ce:6e:8b:24)
  Sender IP address: 192.168.0.114 (192.168.0.114)
  Target MAC address: 00:00:00:00:00:00 (00:00:00:00:00:00)
  Target IP address: 192.168.0.1 (192.168.0.1)
```

The packet details window of the first packet in the capture file is very straightforward. The computer at 192.168.0.114 needs to communicate with the computer at 192.168.0.1, but doesn't know its MAC address. Notice that the target MAC address here is 00:00:00:00:00:00. This being the case, it sends a packet with the destination address ff:ff:ff:ff:ff:ff, in turn broadcasting that packet to everything on the current network segment. This is the basic ARP Request packet, as stated in the Opcode field.

Step 2: Computer B Receives the Request and Broadcasts an ARP Reply Packet

```
Frame 2 (46 bytes on wire, 46 bytes captured)
Ethernet II, Src: D-Link_0b:22:ba (00:13:46:0b:22:ba), Dst: HonHaiPr_6e:8b:24 (00:16:ce:6e:8b:24)
Address Resolution Protocol (reply)
  Hardware type: Ethernet (0x0001)
  Protocol type: IP (0x0800)
  Hardware size: 6
  Protocol size: 4
  Opcode: reply (0x0002)
  Sender MAC address: D-Link_0b:22:ba (00:13:46:0b:22:ba)
  Sender IP address: 192.168.0.1 (192.168.0.1)
  Target MAC address: HonHaiPr_6e:8b:24 (00:16:ce:6e:8b:24)
  Target IP address: 192.168.0.114 (192.168.0.114)
```

The second packet is our reply from 192.168.0.1. This device received the ARP Request in step one, and generated this reply addressed to 192.168.0.114. Notice that this reply contains the information that 192.168.0.114 needs to communicate properly. This is the sender MAC address in this second packet. You can tell immediately that this is an ARP reply by looking at the Opcode field.

Step 3: Communication Can Begin

Once the device at 192.168.0.114 receives the ARP Reply it can then take the MAC address of 192.168.0.1 and put it in its ARP table for future use. With this new information, ARP can successfully translate between layer two and layer three so that communication can move on to the physical medium.

Homework

ARP is by far one of the simpler protocols you will see, which is why I chose to use it for our first session of Packet School 201. Take a packet capture of your network and you are bound to see some ARP packets flying across the wire every now and again. See if you can

pinpoint some of these and isolate the requests and replies. If you really want to learn a bit more about ARP, and how it can be used for malicious purposes, do some reading on ARP cache poisoning.