5

ONE NIGHT ON MY ISP

So far, I've used visualization to help analyze network reconnaissance, vulnerability assessment, and network attack. While these examples were useful in isolation, they didn't fully address the problems you may face when visualizing live network traffic. In this chapter I'll bring together many of the visualization techniques presented thus far and use them to analyze more complex and difficult-to-predict activity. To do this, I'll use a dataset from a home ISP connection, and I'll analyze it using the RUMINT visualization tool (http://www.rumint.org), as well as the Wireshark protocol analysis tool (http://www.wireshark.org). The purpose behind using such a dataset is to get a feel for the type of malicious behavior that constantly occurs on the Internet. I'll also take the opportunity to illustrate the use of visualization to analyze activity from a wide range of sources.

Analyzing the ISP Dataset

Sometimes called *Internet background radiation*,[1] network traffic from compromised systems and misconfigured or broken network devices is common on the Internet. Because of this continuous stream of traffic, much of which comes from network worms, a typical, unpatched home computer only has hours, sometimes minutes, before it is successfully compromised.[2] In some cases the compromise will occur so rapidly that the user wouldn't even have time to connect an unpatched system directly to the Internet and download the latest patches before that system was compromised. To better understand the type of malicious traffic a home computer is subject to, I'll use visualization to analyze *unsolicited* network traffic. In other words, the questions I'm seeking to answer are what type of malicious traffic does a home network receive on a typical day, and how is it malicious? For this experiment, I connected a single machine to a cable modem and captured incoming packets, without transmitting any.[3] With the legitimate network disconnected, I let the computer collect packets for a nine-hour period through the night, gathering 303 packets.

Big-Picture Analysis

At this point we have an unfamiliar dataset, so let's begin the analysis by trying to gain a big-picture overview of what it contains. I'll start by opening the dataset in the parallel coordinate plot view, shown in Figure 5-1. This allows us to see the general behavior of all the packets in one screen. I've assigned individual colors to common protocols: TCP is green, UDP is orange, and ICMP is blue. For all other protocols, including ARP, I've used yellow. Due to the wide range of packet types from across the Internet, the display is a bit messy. However, there are still some general trends we can detect and use to help guide our analysis. In particular, you can easily determine which header fields are constant, and when dealing with more values, you can see the distribution. By watching the animated playback of the packet capture, you can also see which values occur sequentially or randomly.

As you examine Figure 5-1, note that I chose what appears to be a somewhat arbitrary ordering of the axes. In actuality, this sequence closely maps to the occurrence of header values in the actual packets. The only exception is the first axis, which displays packet length. This value is added by pcap-compliant packet capture programs and contains the length of the packet as seen by the capturing device. Subsequent axes in the plot come from header fields. I believe this sequence is useful for learning about general packet structure and understanding the behavior of individual fields. Later in the book I will use other sequences that highlight interesting correlations in the data.

[1] For more information on Internet background radiation, see Ruoming Pang's "Characteristics of Internet Background Radiation" from the proceedings of the 4th ACM SIGCOMM Conference on Internet Measurement (Taormina, Italy, 2004), pp. 27–40.

[2] See the Honeynet Project's trend analysis at http://www.honeynet.org/papers/trends/life-linux.pdf for more details.

[3] While this machine didn't transmit any packets, the cable modem did, as we will see later in the chapter.

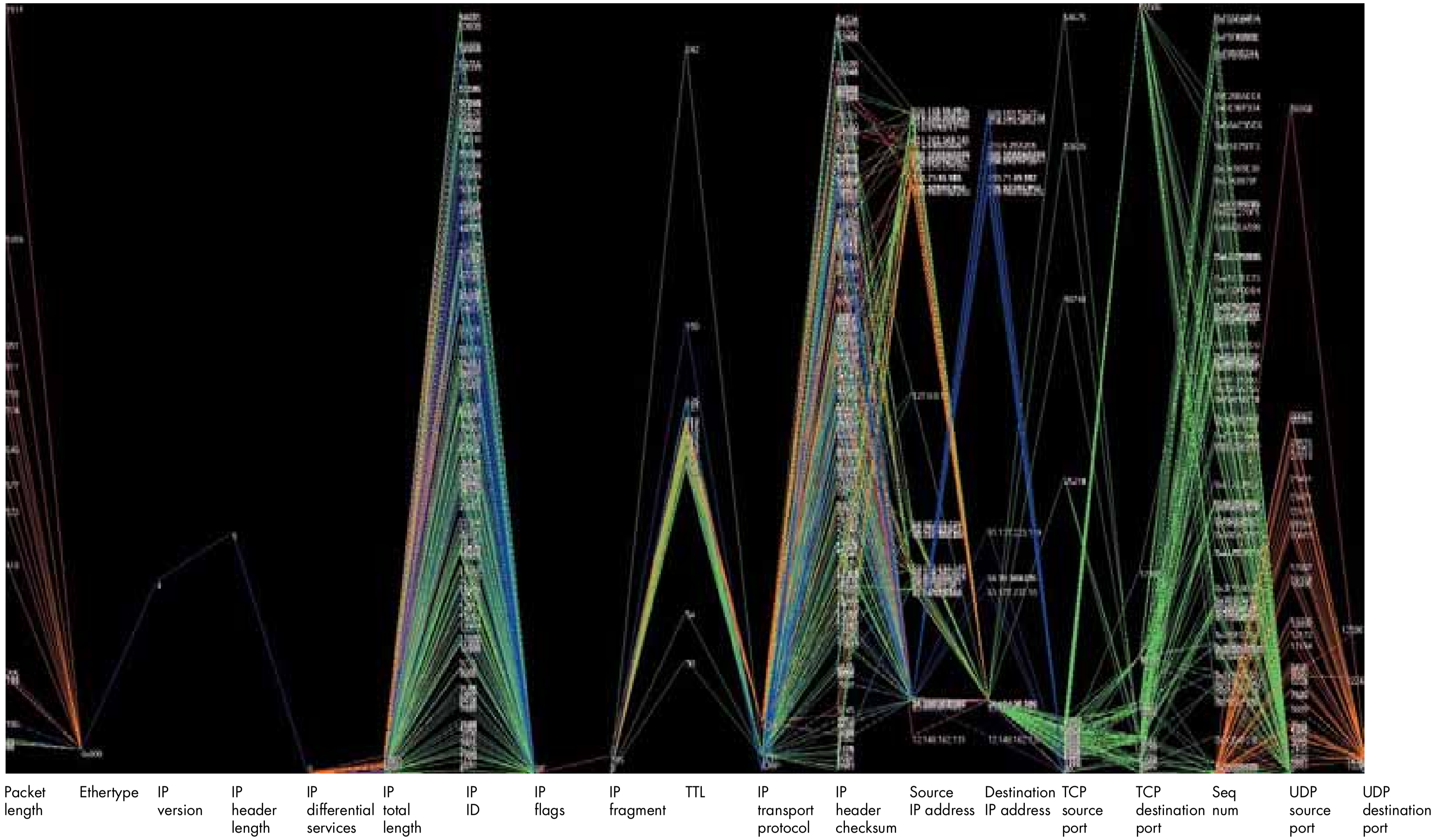


Figure 5-1: A big-picture overview of the entire home ISP dataset. Despite the noise, we can still detect general trends that can help guide our analysis.

Similarly, the scale of the axes varies significantly and reflects the total range of possible values as determined by the number of bits allocated in the packet header structure. For example, the IP header uses 16 bits as a checksum, so the corresponding axis scales the range of possible values, 0 to 65,535 (2^{16}), to the axis. I believe this is a reasonable starting point for each of the axes, but I will alter several axis scales later in the chapter to help reveal useful detail.

Ordering the axes to mirror the sequence in which header fields occur in the packet structure leads to an important point. Visualization, particularly when combined with a tool like Wireshark, is great for learning about the structure of packets as well as how network protocols operate. If you haven't done so, I suggest you download RUMINT and Wireshark and begin observing network traffic. If you are interested in exploring how networks behave, your time will be well spent. With that said, let's examine each of the header fields.

Packet Length

If you examine the first column of Figure 5-1 (packet length), you will notice that all the traffic, except for UDP packets (in orange), is very small—less than 100 bytes per packet. Because similar packets will take identical paths in portions of the plot, I confirmed this observation by filtering all the UDP packets to verify that none of the remaining packets were larger than 100 bytes. You can see the distinction quite clearly in the packet length visualization in Figure 5-2. Recall that TCP requires a three-way handshake to create a communication session; because no machines were operating on the internal network, the remote computers were unable to create a session or send the payloads they intended to send. UDP, on the other hand, does not require session establishment and can contain the payload in a single packet.

Most likely, these TCP packets were either generic port scans or probes to check for vulnerable services. Often, if an attacker detects a vulnerable service, he establishes a session, exploits the vulnerability, and then

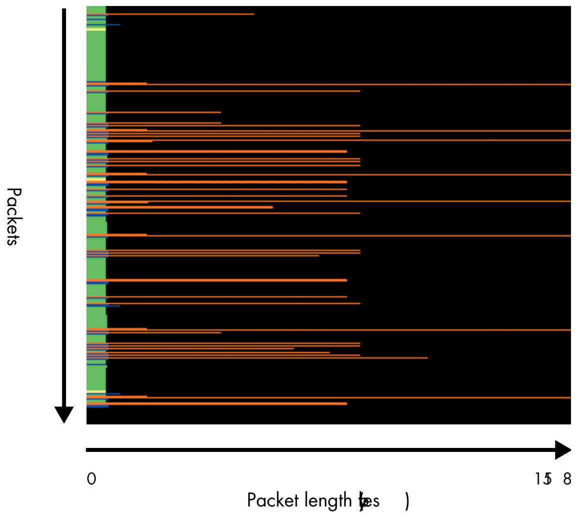


Figure 5-2: Comparison of the lengths of all packets in the dataset. Note that the UDP packets (orange) are far larger than the TCP (green), ICMP (blue), and ARP (yellow) packets.

transfers a malicious payload of some sort. The attacking program won't send the payload without session establishment, as is the case in this example, but tricking these programs into sending the payload can be quite valuable for analysis. This requires using a listening program that behaves similarly to a vulnerable service in order to convince the attacking software to transfer the payload.[4]

Ethernet and IP Versioning Information

Now that we've taken a close look at the first axis, packet length, let's examine the next four columns of Figure 5-1 for useful insights. (A detailed view of these columns is shown in Figure 5-3.) The first, *Ethertype*, is an Ethernet header field that describes the type of packet it is carrying. For IP packets, this field is a hexadecimal value of 0x800, but notice the label in the figure—it contains what appears to be another similar value, probably 0x806 for the Address Resolution Protocol (ARP). The impact of my initial scaling decision is evident in this image. At first glance, it appears that the value is 0x808. To confirm my assumption, despite potential labeling occlusion, I used RUMINT's packet slider bar to step through the packets one at a time (Figure 5-4, right), and I found the first ARP packet in the dataset (Figure 5-4, left). To address this kind of labeling occlusion, I plan to autoscale each axis in future versions of RUMINT. Instead of scaling each axis based on the range of possible values in each header field, I'll scale the axis based on the range of values actually present in the dataset.

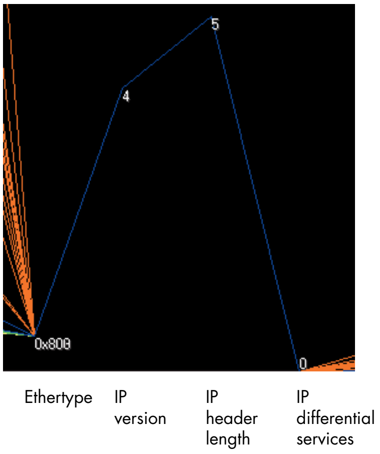


Figure 5-3: Detail view of the Ethertype, IP version, IP header length, and IP differential services fields

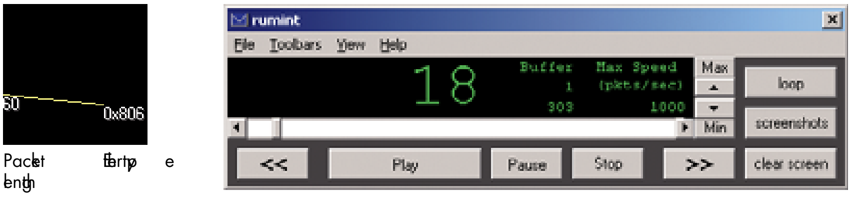


Figure 5-4: Using RUMINT's slider bar to locate the first ARP packet (left) at position 18 in the dataset (right)

[4] For more on malware collection, see Paul Bächer, Thorsten Holz, Markus Kötter, and Georg Wicherski's "Know Your Enemy: Tracking Botnets" white paper at http://www.honeynet.org/papers/bots, as well as the Malware Collection portal at http://www.mwcollect.org.

The remaining three fields in Figure 5-3 are also useful. The figure tells us that, for the entire dataset, all the packets are IP version 4 and have standard-length headers. The constant (zero) value in the *IP differential services field*, sometimes called the *Type of Service (TOS) field*, tells us that every packet was regular precedence and had routine delay, throughput, reliability, and cost settings.[5] Variations here, such as the presence of IP version 6 or higher-priority precedence packets, could indicate more subtle attacks.

IP Fragmentation

Now let's look at the next set of fields, those shown in Figure 5-5. *IP total length* indicates the length of the IP packet, which is the size of the IP header and the payload of the IP packet. This value differs from the previous packet length example, which also included the Ethernet header and trailer fields. We can see that the lengths here are all small; very large packets might indicate an attempt to crash systems or other types of attack. One example is the now uncommon ping of death. Attackers found that they could crash machines by sending an IP packet larger than the maximum legal IP packet size of 65,535 bytes.[6] Packets of up to 65,535 bytes are supported by IP, but some link-layer protocols between the sender and the destination are unable or unwilling to handle such large packets, usually for network performance reasons. Because of this, IP allows large packets to be fragmented into smaller ones and reconstructed at the destination. IP fragment reconstruction is often performed differently on different platforms, and attackers have exploited this fact to avoid being discovered by intrusion detection systems.[7] Therefore, fragmented packets, particularly those that are unsolicited, should raise suspicion.

The next fields provide IP the information it needs to perform fragmentation. The *IP identification field* is used to uniquely identify the fragments belonging to a given IP packet. Figure 5-5 shows a range of IP identification values; based on my experience, no anomalies jump out of this range. The next field, *IP flags*, typically contains two types of control flags: don't fragment (DF) and more fragments (MF). *DF* indicates whether the packet should be fragmented, and *MF* indicates whether more fragmented packets are en route. Similarly, the *IP fragment field* is used by the receiving computer to correctly order and reconstruct the arriving fragments. Because the values for MF and DF are very similar (2 and 4, respectively), RUMINT places them close together on the axis; this makes it difficult

[5] For more on these and other parameters, see http://www.networksorcery.com/enp/protocol/ip.htm.

[6] You can find out more about the ping of death at http://insecure.org/sploits/ping-o-death.html.

[7] For more information on this class of attack, see Thomas H. Ptacek and Timothy N. Newsham's "Insertion, Evasion, and Denial of Service: Eluding Network Intrusion Detection" (http://insecure.org/stf/secnet_ids/secnet_ids.html).

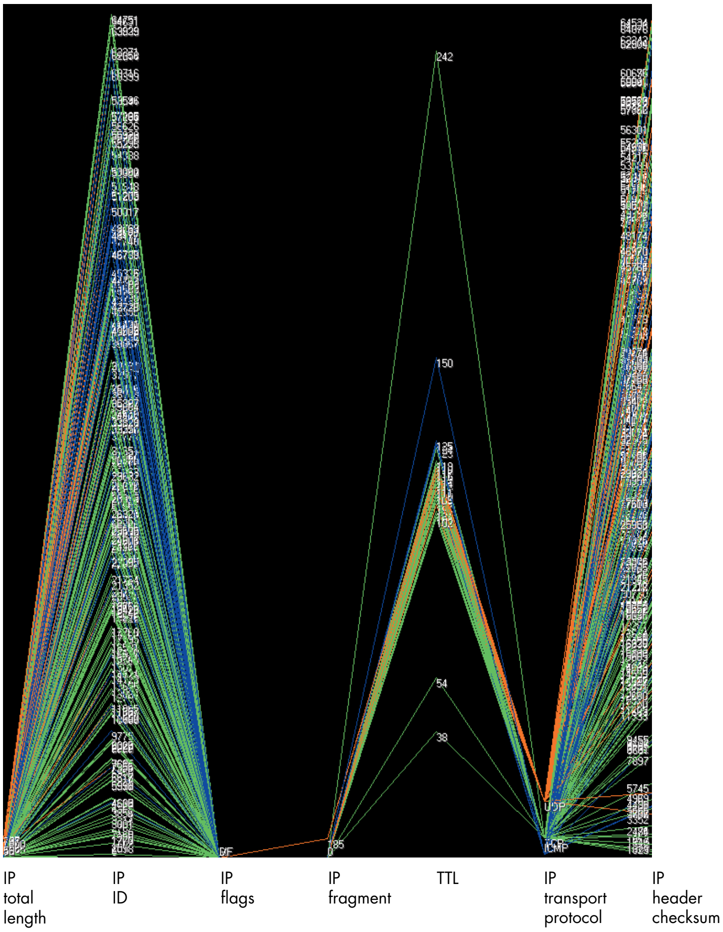


Figure 5-5: The next set of header fields

to distinguish between the two labels. To verify the existence of both MF and DF in the dataset, I use the slider bar to identify individual packets of each class (Figure 5-6). This tells me to be on the lookout for suspicious fragmentation behavior.

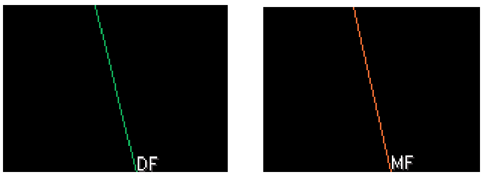


Figure 5-6: Detail view of the IP flags axis. I use the packet slider on the VCR-like interface to quickly identify fragmented packets in the dataset.

TTL, Transport Protocols, and Checksums

The remaining three fields in Figure 5-5 show the distribution of TTL values, the transport protocols used, and the IP checksums. As I mentioned earlier in the book, *TTL values* are used to prevent misrouted packets from cycling endlessly across the Internet. A packet's TTL value is decremented by one as it makes each hop across the network. Here we can see a cluster of activity from approximately 100 to 125. Also note the outliers: 242, 150, 54, and 38. These might be indicative of special attack classes and may be worthy of additional inspection. This example illustrates the strength of visualization—at a glance, you are able to detect the clusters and the outliers, which would be difficult to do using textual tools.

The *IP transport protocol field* is relatively straightforward; it contains a value between 0 and 255 that indicates the type of transport protocol contained within the IP packet. RUMINT maps this numeric value to the appropriate text label. For example, a value of 1 indicates ICMP, 6 indicates TCP, and 17 indicates UDP. This is an example of augmenting the packet capture dataset with external semantic data to provide a richer, more intuitive display for the user. If you look at Figure 5-7, note that you see three types of transport layer protocols: ICMP, TCP, and UDP. RUMINT places values on this axis according to their position on the possible range from 0 to 255. Because these protocols' numeric values are all similar, the result is difficult to read. A possible future enhancement would be to make wiser use of the entire vertical axis, perhaps dividing the axis by the number of protocols and allocating the space accordingly. The result would be significantly greater separation and no occlusion, unless many different protocols were present. I wasn't 100 percent certain that another protocol wasn't hidden in the noise, so I filtered TCP, UDP, and ICMP, but I found no other protocols. Note that ARP, while present in the dataset, is of Ethertype 0x806 and doesn't use the IP (0x800) packet structure, so it is not present in the IP transport protocol field. From the analysis thus far, we know that the dataset only contains TCP, UDP, ICMP, and ARP.

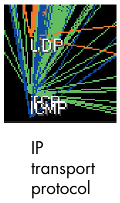


Figure 5-7: Detail view of the IP transport protocol field

The final field in this set is the *IP checksum*, a calculation designed to detect errors in the IP header and options information.[8] If some portion of the IP header changes, even by one bit, the IP checksum will not match when a network router or the destination computer recalculates it; this may cause the packet to be discarded. The IP checksum column doesn't tell us much about our dataset, except that we see values from across the possible range. Packet checksums are worth considering, however, since invalid checksums may indicate foul play. A number of attack tools I've encountered do not generate correct checksums when they create packets. It is easy

[8] *IP options* are optional extensions to the IP header; you can find out more about them at http://www.iana.org/assignments/ip-parameters.

enough to determine whether there is a problem with a checksum, because you can just calculate what it *should* be based on the packet's contents and then compare that value to the value in the IP checksum field. Wireshark does this automatically and reports invalid checksums; you could do the same in a visualization system.

IP Addresses, Ports, and Sequence Numbers

The remaining fields contain the source and destination IP addresses, TCP and UDP ports, and TCP sequence numbers. Notice in Figure 5-8 that I've

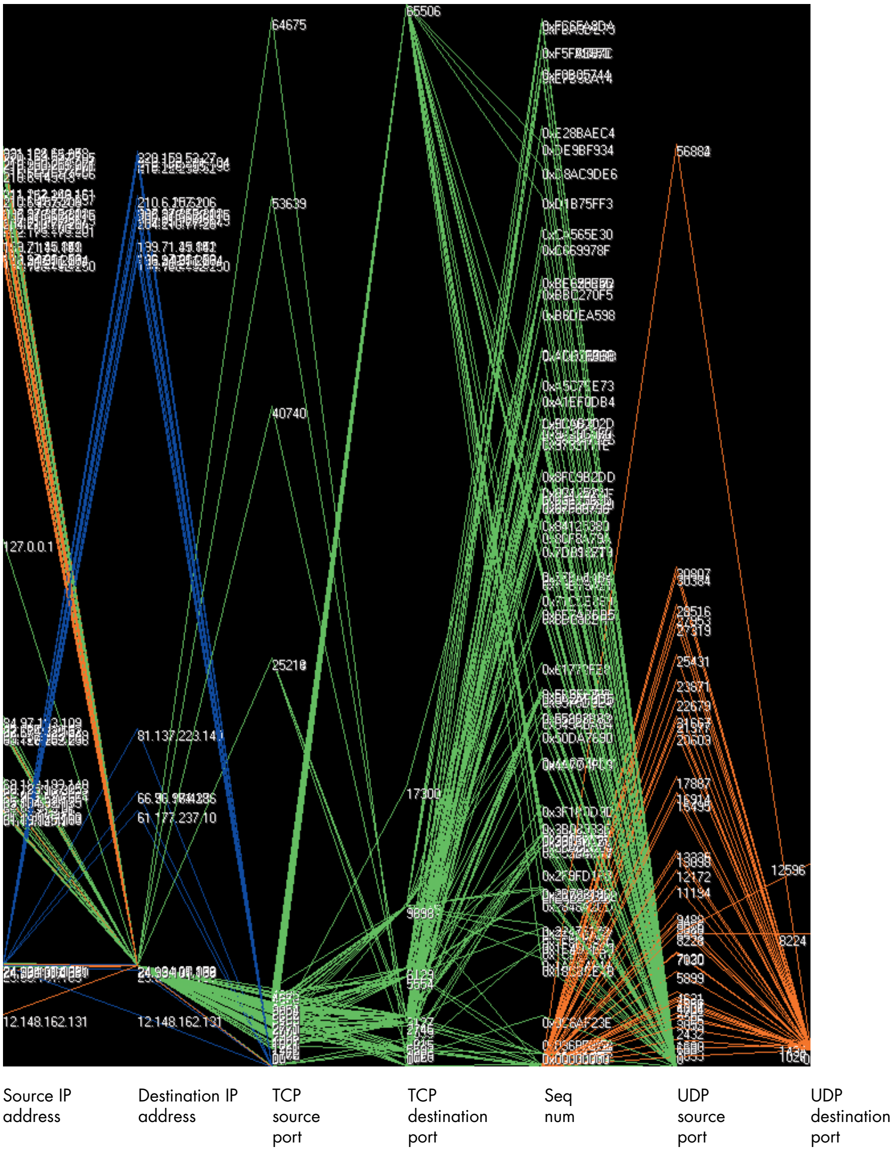


Figure 5-8: IP addressing and port information

separated the TCP and UDP's source and destination ports. I made this design decision to more easily allow independent analysis of TCP and UDP port activity. Alternatively, you could combine TCP and UDP into a single pair of source and destination axes. You would save screen space, but you would also have a more occluded display, because port activity for the two protocols might overlap. Despite my separation of the TCP and UDP's port activity, the image is still rather messy, so lets try to clean things up a bit. Because I've already confirmed that we have only ARP and three IP transport layer protocols in the dataset (TCP, UDP, and ICMP), I'll focus my analysis by examining these protocols one at a time.

Analyzing and Removing Slices of Traffic

When confronted with a new dataset for analysis, the first step is to gain big-picture context of what it contains, and we did just that in the previous section. Now our goal is to gain a deeper understanding of the data. The general approach I'll use is that of selectively analyzing and then removing similar slices of network traffic. Let's start by examining each of the four protocols in isolation.

TCP

By removing the extraneous protocols and focusing on the packet lengths, source IP addresses, and destination ports of only TCP packets, we have a far cleaner imager. However, what I left out is just as important a decision for analysis as what I left in. I removed the destination IP address field because it was the same for all packets (i.e., the address of the home ISP connection); the UDP source and destination ports, as they did not apply to this protocol; and the TCP sequence numbers, because I did not believe they contained any useful insights in this situation.[9] I chose to include packet length to reconfirm that all the TCP packets were small, which is an indicator that they are probably simple probes. I am reconfirming my observation because there could have been larger TCP packet lengths that were obscured by similarly sized packets of other protocols that were plotted later, and thus on top of, TCP packets that were plotted earlier. I also included source IP addresses to see which IP addresses were claiming to send the packets, and most importantly for this analysis, I included TCP destination ports to see which services were being targeted. Figure 5-9 shows that all the

[9] It isn't that TCP sequence numbers aren't relevant in other circumstances. For more on security implications of sequence numbers, see Michal Zalewski's "Strange Attractors and TCP/IP Sequence Number Analysis" at http://lcamtuf.coredump.cx/newtcp.

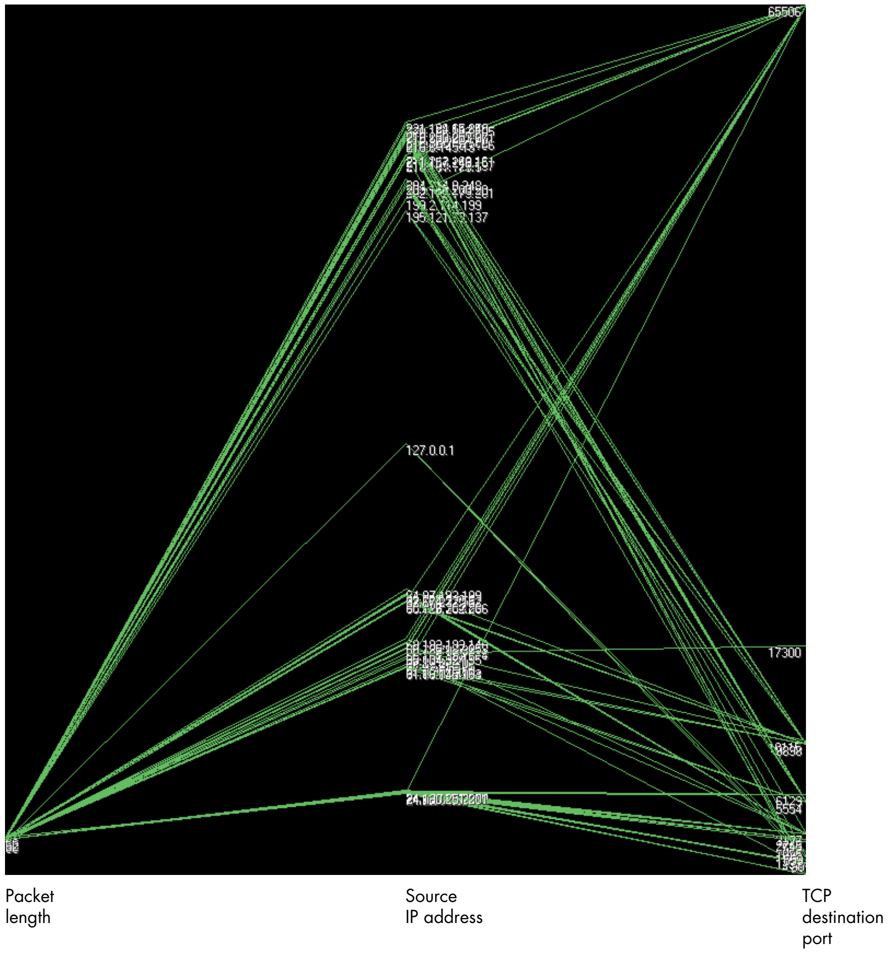


Figure 5-9: By filtering the ISP dataset to show only TCP packets and carefully choosing the fields to display, we are able to generate a far clearer picture.

packets were indeed small (less than 100 bytes) and that several clusters of Internet addresses were performing the probes against a small number of ports.

One drawback of the parallel coordinate plot view is its poor ability to accurately display the quantity of packets sent.[10] I'd like to know which ports were probed most frequently, so I'll switch to the combined visualization

[10] There is a variant of the parallel coordinate plot that uses a density histograph, sometimes called an *axis histogram*, which helps overcome this limitation.

you saw earlier in the book. Figure 5-10 shows the result. Notice in the right panel that the low-numbered ports were most heavily targeted but that there were two high-numbered outliers, ports 65506 and 17300. I'll make a note of these two ports and continue with my analysis. I'd like to know all the ports that were targeted, so I'll zoom in further.

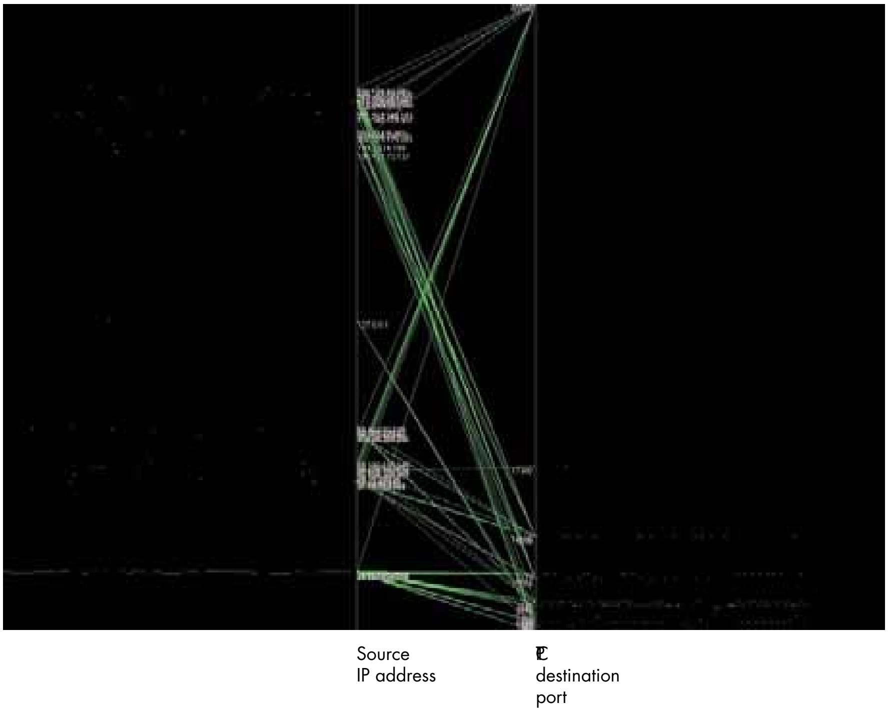
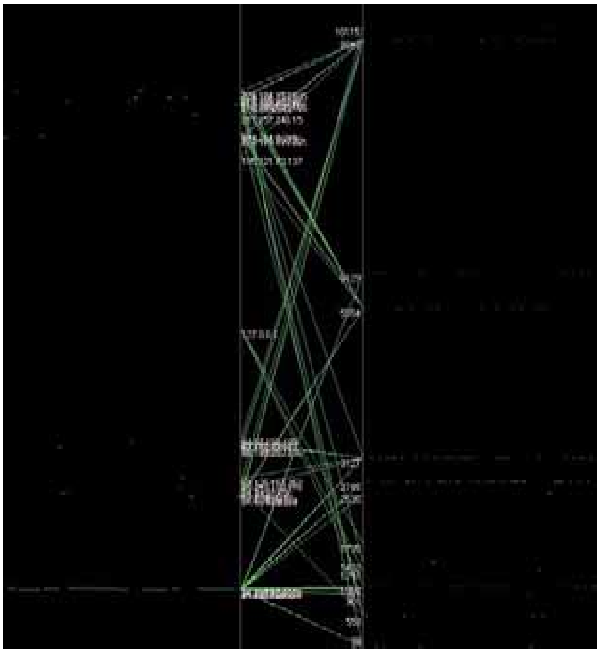


Figure 5-10: The combined visualization overcomes the parallel coordinate plot's weakness by showing the number of packets initiated by each IP address as well as the number that targeted each port.

By examining the two images in Figure 5-11 (and revisiting Figure 5-10), we can now build a complete list of the 17 TCP ports that were probed during the night. As you examine the images, note that several destination ports received significantly more traffic than the others: 1025, 2745, and 3127. We'll take a look at these ports in a moment.



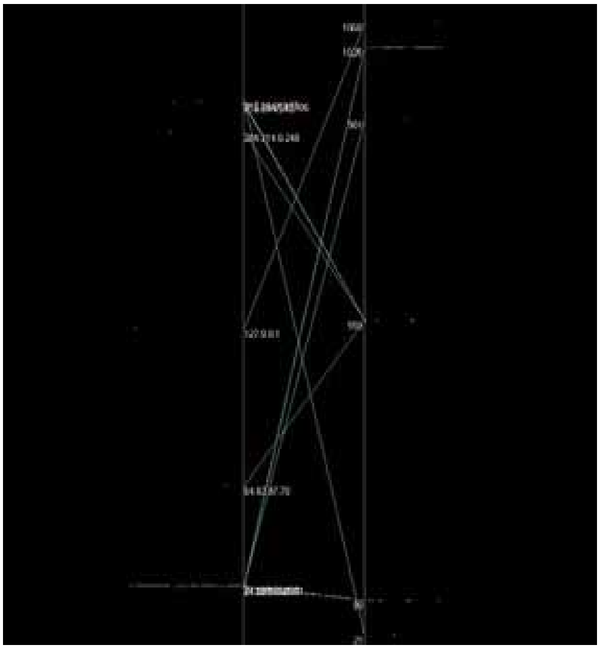


Figure 5-11: By zooming in we can clearly see which ports were being targeted and how heavily. The top image is zoomed in to show only packets below 10500. In order to clearly see the remaining low-numbered ports, the bottom image is zoomed in to show ports 1100 and below.

Table 5-1 shows all of the TCP ports that were targeted, as well as the number of packets they received. By taking a look at the table, you can confirm that ports 1025, 2745, and 3127 did indeed receive significantly more traffic. In Figure 5-12, I've placed the same data on a bar chart and sorted the columns according to activity. It is important to note that the parallel coordinate plot and combined visualization helped provide a big-picture overview of the port activity as well as insight into general port behavior, but these visualizations could always be combined with the more traditional textual table and bar chart to facilitate in-depth analysis. Neither RUMINT nor

Table 5-1: Targeted TCP Ports in the Dataset

Port	Notes	Number of packets
21	FTP	1
80	HTTP	12
559	Domwis and Solufina trojans	4
901	SMPNAMERES and Samba Web Administration Tool	1
1025	Microsoft Remote Procedure Call (RPC) service	32
1068	Sasser worm	1
1293	PKT-KRB-IPSec	1
1433	Microsoft SQL Server	2
1735	PrivateChat	1
2535	Bagle and Beagle viruses	3
2745	Bagle, Beagle, and Tanx viruses	47
3127	Mydoom virus	30
5554	Sasser and Dabber worms	12
6129	Dameware remote administration server	16
9898	Sasser	13
10115	Ganymede Endpoint	1
65506	Virus proxy servers, spam email relay	21

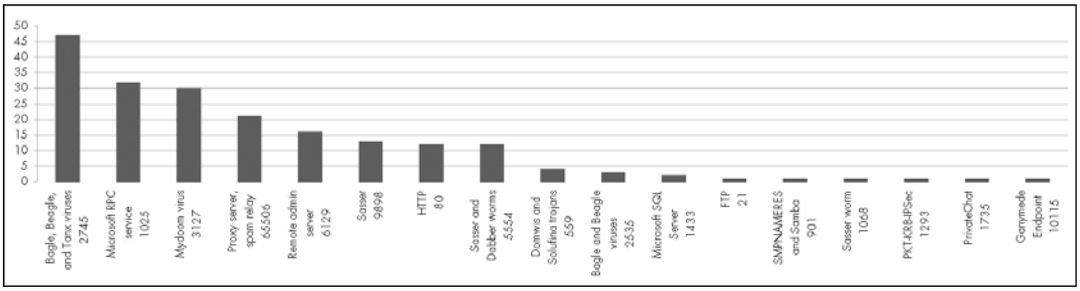


Figure 5-12: Number of packets sent to TCP ports

Wireshark has the ability to easily generate the information in Table 5-1 and Figure 5-12. I created both by manually inspecting port activity in the visualization and using Wireshark to count individual packets, a tedious process. I recommend that if you design your own systems, you look for intelligent opportunities to combine visualization with other, more traditional analysis techniques.

Let's take a look at the most active ports, but before we do, it is important to note that packet volume alone doesn't necessarily indicate the severity of an attack. For example, the most active ports could have been targeted by a new network worm outbreak, or all of that activity could be remnants of some previous occurrence for which your machines have already been patched. Low-volume activity could indicate a newly discovered exploit and potentially be very dangerous. Alternatively, this same sort of traffic could be harmless backscatter from malicious activity occurring elsewhere on the Internet or from misconfigured network devices. Whether you choose to focus on high-volume activity, low-volume activity, or both depends on your instinct and experience as an analyst. Here I'll focus on the high-volume ports because I believe they will help provide useful examples for analysis.

Port 2745 was the most heavily targeted port, and it is used by the Bagle mass email virus, which first surfaced in early 2004. After a host is compromised, Bagle sets up an SMTP engine that it uses to propagate, and then it installs a back door on port 2745 that allows an attacker to upload and execute files.[11] Activity on this port is likely to be scanning by other attackers looking for that back door on infected systems. If the host computer had sent a response, we would have probably seen a series of packets trying to connect and control the machine.

Port 1025 is used by the Microsoft Remote Procedure Call (RPC) service. *RPC* is intended to assist distributed applications development by allowing a program on one machine to execute a procedure on a remote machine. Unfortunately, the Microsoft RPC service has been plagued by vulnerabilities. The activity on port 1025 is almost certainly a series of probes for vulnerable RPC services. The activity on the third most active port, 3127, is similar to the activity on port 2745. The Mydoom virus sets up a back door on this port; as a result, attackers have developed a number of network worms to exploit the presence of the back door, including Doomjuice, Welchia, and Deadhat.

I'll stop my TCP analysis here, but we could dig deeper into the individual packets looking for other insights, continue to research known activity targeting the ports, or set up responders to capture malware. If this were your computer, you'd certainly want to make sure that its patches were up to date and that your firewall was appropriately configured. However, for our purposes, the key idea is that visualization allowed us to quickly determine the nature of the TCP packets, and once we did, it enabled us to move on and explore other activity.

---

[11] See http://www.linklogger.com/TCP2745.htm for more information.

UDP

As we saw earlier, the UDP packets were far larger than the TCP packets and, because of this, I suspect that they will contain some interesting packet payloads. To regain our bearings, let's look at a parallel coordinate plot of the UDP packets, which is shown in Figure 5-13. I've zoomed in slightly, since there were no UDP destination ports above 12596. We can see that ports 12596, 8224, and 1434 all received traffic; there is an interesting cluster in the 1020s that received the widest variation of packet sizes. I've included a detail view in Figure 5-13 in which I've changed the scale of the UDP destination port axis to see this cluster more clearly; we can now identify the ports as 1026, 1027, 1028, and 1029. Figure 5-14 uses a combined visualization to get a feel for the number of packets directed to each UDP port. Ports 1026 through 1029 have received the most traffic, by far. Now that we know the basic port activity, let's take a closer look at each port.

Ports 12596 and 8224 bring up an interesting learning point. As I searched security websites for these ports, I was not able to find anything of note. Suspicious, I opened Wireshark and created a filter that would display only these ports. When I applied the filter, Wireshark did not display any packets. Even more curious, I returned to RUMINT and used its port filters to help identify the packet number of one of the packets, and then I returned to Wireshark and examined that packet. Aha! The packet was reported by Wireshark as fragmented IP, as were the other packets sent to these ports. The IP header reported these packets as UDP, but because they were fragmented, RUMINT incorrectly attempted to identify specific UDP ports. This example demonstrates Wireshark's clear strength as a protocol analyzer—it understands a comprehensive list of protocols. By using a complementary approach, switching back and forth between the two tools, I was able to exploit the protocol-processing strengths of Wireshark and the rapid-analysis capabilities of visualization.

Next, I turned to the three packets targeting port 1434, which is typically used by Microsoft SQL Server. To better observe these packets, I set RUMINT's filtering capability to display only packets destined for UDP port 1434. By looking at the parallel coordinate plot view, I could tell that all three packets were the same size, 418 bytes, but they came from three significantly different IP addresses. I used the American Registry for Internet Numbers (ARIN) IP address lookup service (http://www.arin.net/whois) and found that two of these addresses were in Europe and one was in Asia.[12] I then opened a detail view window and played the traffic again.

[12] These locations may or may not be accurate; it isn't difficult to spoof a source IP address using UDP.

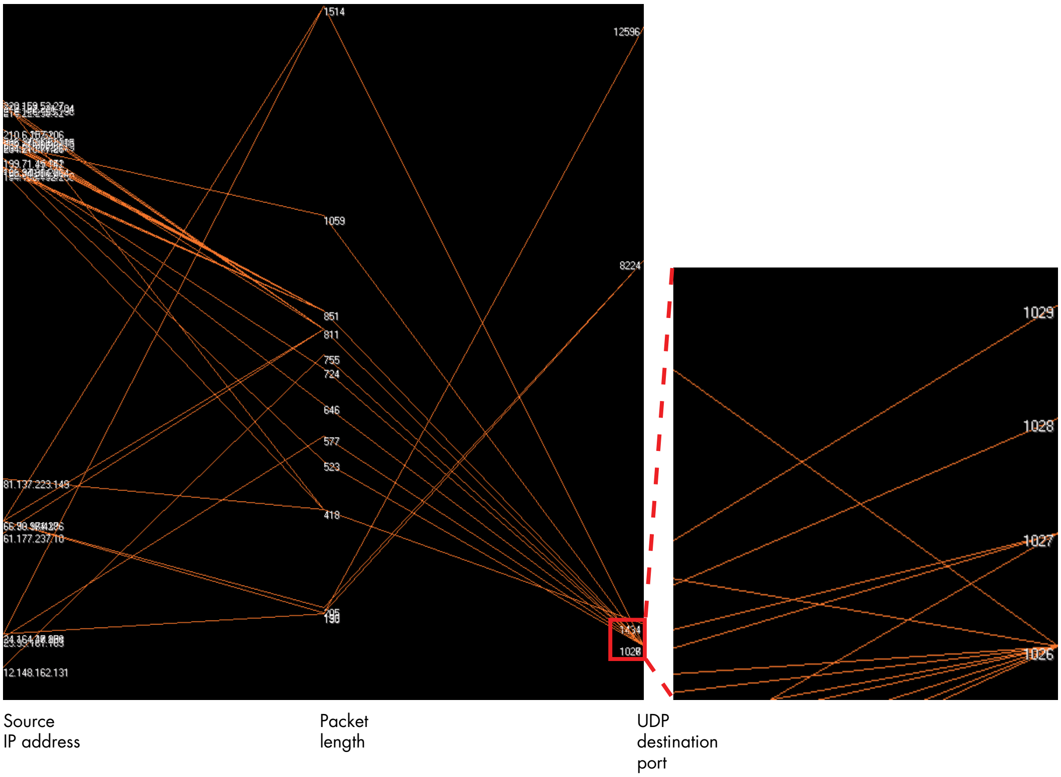


Figure 5-13: After examining the TCP packets, I used a parallel coordinate plot to reorient the UDP analysis. In the detail view, I've changed the axis scale to identify the specific ports in the 1020s.

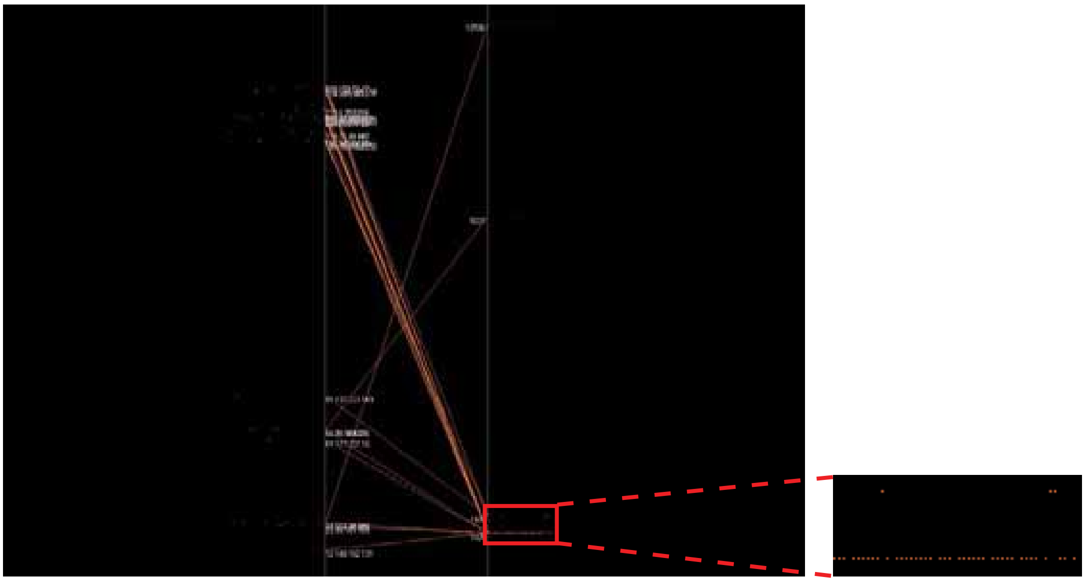
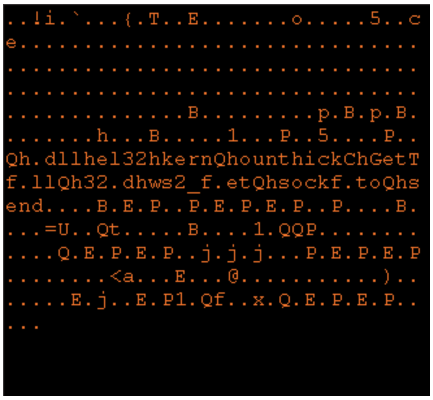
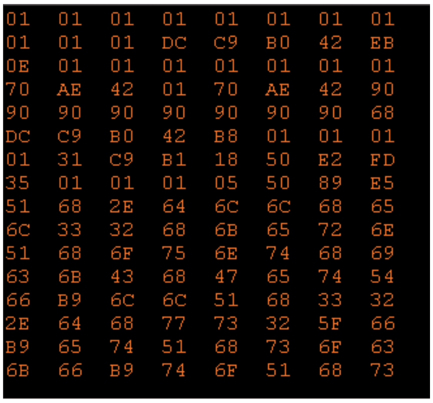


Figure 5-14: Using a combined visualization to see the number of packets directed to each UDP port. Notice that ports 1026 through 1029 were most active.

During this playback, I could see that all three packets carried the same payload. Figure 5-15 shows one of three packets' contents. The image on the left shows only the printable ASCII characters and uses periods for all other values. The image on the right shows the actual byte values. Note the distinct textual contents in the image on the left. A quick Internet search for this text shows that this is the SQL slammer worm, which exploited a buffer overflow in Microsoft SQL Server to rapidly spread across the Internet. Unlike the small TCP packets we looked at earlier in the chapter, this UDP packet contains the code the slammer needs to infect susceptible systems and continue to spread.[13]



```
..!i.`...{.T..E.......o.....5..c
e...............................
................................
................................
..............B.........p.B.p.B.
.......h...B.....1...P..5....P..
Qh.dllhel32hkernQhounthickChGetT
f.llQh32.dhws2_f.etQhsockf.toQhs
end....B.E.P..P.E.P.E.P..P....B.
...=U..Qt.....B....1.QQP........
....Q.E.P.E.P..j.j.j...P.E.P.E.P
........<a...E...@..........)..
.....E.j..E.P1.Qf..x.Q.E.P.E.P..
...
```



```
01  01  01  01  01  01  01  01
01  01  01  DC  C9  B0  42  EB
0E  01  01  01  01  01  01  01
70  AE  42  01  70  AE  42  90
90  90  90  90  90  90  90  68
DC  C9  B0  42  B8  01  01  01
01  31  C9  B1  18  50  E2  FD
35  01  01  01  05  50  89  E5
51  68  2E  64  6C  6C  68  65
6C  33  32  68  6B  65  72  6E
51  68  6F  75  6E  74  68  69
63  6B  43  68  47  65  74  54
66  B9  6C  6C  51  68  33  32
2E  64  68  77  73  32  5F  66
B9  65  74  51  68  73  6F  63
6B  66  B9  74  6F  51  68  73
```

Figure 5-15: By using a textual view to examine the UDP port 1434 packets in rapid succession, you can see that all three packets have the same characteristic textual strings.

The remaining packets were in a cluster from ports 1026 to 1029. A quick check of the IANA port numbers list (http://www.iana.org/assignments/port-numbers) doesn't tell us much. Port 1026 is assigned to a calendar protocol, port 1027 is unassigned, port 1028 is listed as no longer in use, and port 1029 is assigned to something called the Solid Mux Server. There's not much help here, so let's apply visualization.

For this next step I'll use a technique I created called byte presence visualization. I plot each packet on a horizontal line and illuminate pixels in the 256 columns along the horizontal axis if the packet contains a byte of the corresponding value. For example, if the third packet in Figure 5-16 contained the uppercase alphabet (byte values of 65 through 90) the third row from the top would have illuminated pixels in columns 65 through 90. In Figure 5-16, I've left in the ICMP packets (blue) because I want to see their relationship with UDP (orange). I did this because during my earlier analysis, it seemed that an ICMP packet immediately followed each UDP packet.

[13] For more on the SQL slammer worm, see http://en.wikipedia.org/wiki/SQL_slammer_(computer_worm).

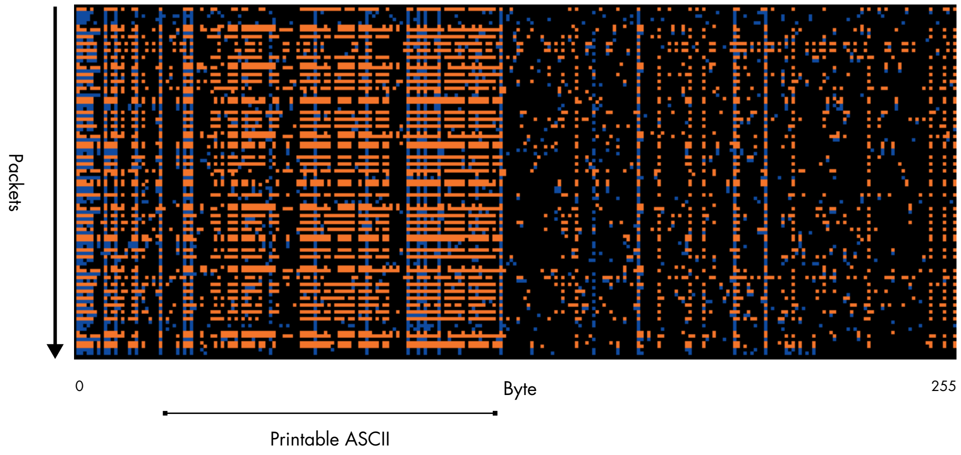
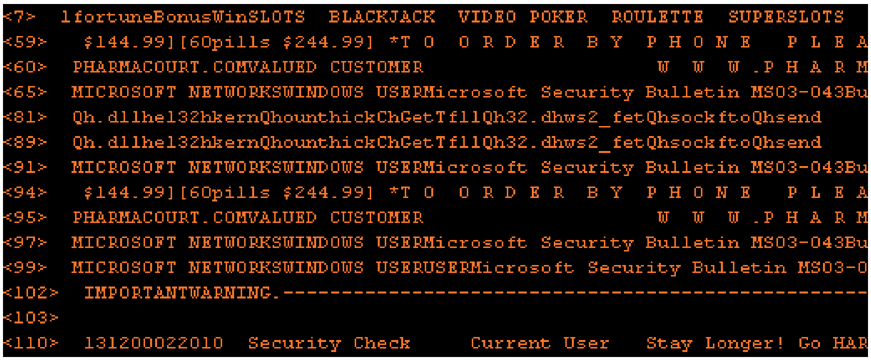


Figure 5-16: Byte presence visualization of the UDP and ICMP packets. The solid orange regions indicate the significant presence of printable ASCII characters.

Notice that each UDP packet uses a wide range of bytes from the printable ASCII range. Based on this observation, let's look at the UDP packet contents in a textual rainfall view. Figure 5-17 shows the text in the packets, one packet per line. Notice that RUMINT color-codes the text consistently with the color scheme assignments we've been using throughout the chapter. The text rainfall view identifies each packet with its number in the dataset to allow individual inspection using RUMINT's detail view or another tool such as Wireshark. I've also directed RUMINT to display only strings of printable ASCII characters at least six characters long. I chose this length somewhat arbitrarily; the aim is to show only those strings within the dataset that are likely to be words.



```
<7>   1fortuneBonusWinSLOTS  BLACKJACK  VIDEO POKER  ROULETTE  SUPERSLOTS
<59>   $144.99][60pills $244.99] *T O  O R D E R  B Y  P H O N E   P L E A
<60>  PHARMACOURT.COMVALUED CUSTOMER                   W  W  W .P H A R M
<65>  MICROSOFT NETWORKSWINDOWS USERMicrosoft Security Bulletin MS03-043Bu
<81>  Qh.dllhel32hkernQhounthickChGetTfllQh32.dhws2_fetQhsockftoQhsend
<89>  Qh.dllhel32hkernQhounthickChGetTfllQh32.dhws2_fetQhsockftoQhsend
<91>  MICROSOFT NETWORKSWINDOWS USERMicrosoft Security Bulletin MS03-043Bu
<94>   $144.99][60pills $244.99] *T O  O R D E R  B Y  P H O N E   P L E A
<95>  PHARMACOURT.COMVALUED CUSTOMER                   W  W  W .P H A R M
<97>  MICROSOFT NETWORKSWINDOWS USERMicrosoft Security Bulletin MS03-043Bu
<99>  MICROSOFT NETWORKSWINDOWS USERUSERMicrosoft Security Bulletin MS03-0
<102>  IMPORTANTWARNING.-----------------------------------------------------
<103>
<110>  131200022010  Security Check     Current User   Stay Longer! Go HAR
```

Figure 5-17: Textual rainfall view of the UDP packet contents. Notice the presence of spam payloads.

After examining Figure 5-17, it is obvious that the UDP packets contain spam. If you search the Internet for the strings contained in the payloads, you'll find that these advertisements are all Messenger spam. By exploiting the Windows Messenger service, which is designed to provide system administrators with a way to rapidly contact online users, attackers can send spam in a new way. In cases such as this one, the attacker sends a single UDP packet to an unpatched system, triggering a pop-up advertisement that appears on the target's screen.

ARP and ICMP

ARP traffic, shown in yellow in this dataset, did not show any suspicious behavior. There were three ARP request packets sent by the ISP's nearest router asking for the cable modem's MAC address. After each query, the cable modem responded with an ARP reply containing the MAC address. Time is another clue that this was legitimate activity. The three exchanges took place at timestamps of 1,169, 15,508, and 29,848 seconds. The 14,340-second interval equates to almost exactly four hours. Such regular timing is something I'd expect of an ISP maintaining contact with its cable modems.

The ICMP activity was more interesting. ICMP is primarily used by the operating systems of networked computers to send error messages.[14] In several figures throughout this chapter, we've seen that ICMP packets were closely interleaved with UDP packets, but we did not see a similar relationship with TCP packets. I suspect this interleaving was caused by the cable modem sending an ICMP packet each time it received a UDP packet. I'll test this intuition by plotting a parallel coordinate plot of the source and destination IP addresses (Figure 5-18).[15] Note the characteristic *X* shape of the plot; it appears that for every UDP packet, there was an ICMP response. RUMINT doesn't parse the contents of ICMP packets, so I'll switch over to Wireshark. By filtering out everything but UDP and ICMP packets, I see that there were 43 UDP and 50 ICMP packets. Of the 50 ICMP packets, 43 were outbound packets of type Destination Unreachable (Port Unreachable). *Destination Unreachable packets* are used by network devices, such as routers, to report their inability to pass incoming messages to destination hosts.[16] By examining the 43 inbound UDP packets and 43 outbound Destination Unreachable packets, Wireshark confirmed that each UDP packet triggered a Destination Unreachable response.

[14] For more on ICMP, see http://en.wikipedia.org/wiki/Internet_control_message_protocol.

[15] Notice the pink lines in Figure 5-18. RUMINT is only using blue and orange for plotting, but when line segments of these two colors occur side by side, our eyes see pink. Clearly, this isn't a desired state, but it's one you might encounter if you develop your own system.

[16] For more on this message, see http://en.wikipedia.org/wiki/ICMP_Destination_Unreachable.

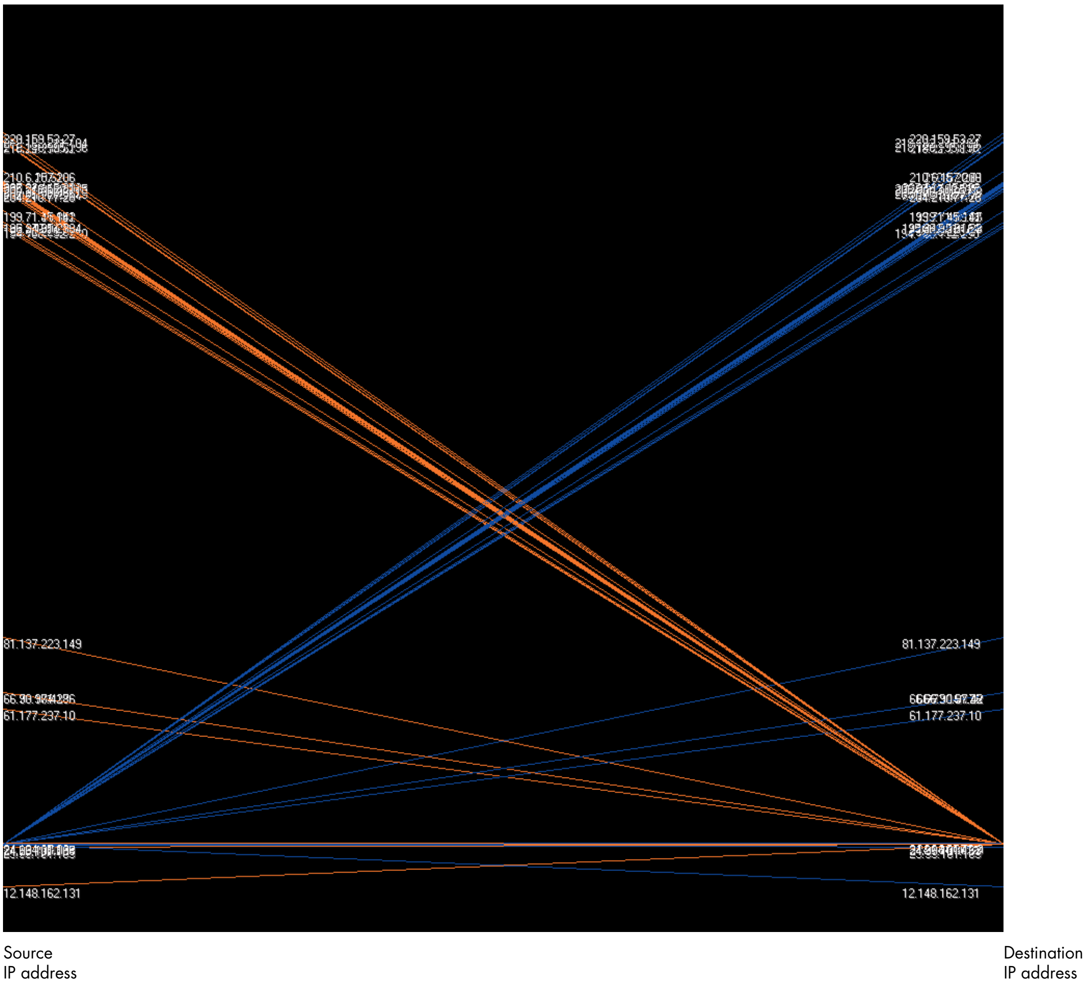


Figure 5-18: Using a parallel coordinate plot to observe the relationship between UDP and ICMP packets. The X shape between the source and destination addresses indicates a possible cause and effect relationship between inbound UDP packets (orange) and outbound ICMP packets (blue).

Finally, while using Wireshark to examine the other seven ICMP packets, I noted that each was an echo request sent to the cable modem. *ICMP echo requests* are used by the ping utility and tools such as Nmap to test for network connectivity. The cable modem did not respond to these requests with an echo reply, probably to help avoid detection by attackers. In this case we see that, while a ping probe won't work, an attacker will get a response if he sends a UDP packet instead, which defeats my stealthy intent.

Conclusions

Previously in this book, we've looked at examples of specific tools in action. While using such noise-free datasets is useful, it masks the fact that real-world network traffic is far messier. In this chapter, I used visualization techniques to rapidly analyze live Internet traffic collected from a home ISP connection. I worked through slices of the traffic based on port and protocol until I had examined all the packets, using Wireshark when its strengths could complement the weaknesses of RUMINT's visualization.

While data analysis of a quiescent home network isn't the most complex security data problem, it is an illustrative one. Much like Honeynet traffic, this relatively small dataset revealed the wide variety of malicious activity occurring on the Internet on a daily basis. If you've ever doubted the importance of securing your network, hopefully this chapter gives you some impetus to make it a priority.

Network sensors are great tools for identifying network incidents—incidents that can then be explored in detail using visualization tools. The sheer number of packets visualized sometimes makes analysis infeasible, however. RUMINT, the tool used throughout this chapter, copes with sensor logs more effectively than other text-based tools, but the displays still become cluttered when they must present datasets of much more than 500 to 1,000 packets. The next logical step in the progression of security data visualization techniques is to find better ways to speed up analysis, despite the presence of high-volume legitimate traffic, perhaps by improving filtering and allowing analysts to create and share their filters.

As you reflect on this chapter, what you should take away are not the specific attacks in this given dataset, as these will certainly change over time. Instead, consider how you can use visualization to support the analytic process, such as facilitating big-picture context, providing details on demand, and progressively working through slices of traffic. If you're so inclined, try capturing your ISP's background radiation and analyzing it in the same way. There are some scary things out there.