Packet Stream Reconstructor (PSR): A Network Security Tool

Nem W. Schlecht
Computer Science
North Dakota State University
Fargo, ND 58105
nem.schlecht@ndsu.edu

Abstract

In today's networks, pattern complexity and packet traffic counts are steadily increasing. Simple packet sniffing of many networks provides too much information in an unorganized and unreadable manner. A solution to this is to analyze this raw packet data and not only provide statistics on the various addresses and ports where communication is occurring, but also to reconstruct any streams of packets into their original format. Such a tool will allow network analysts and security officers the ability to easily see the content, in its human viewable form, of the traffic that is occurring on their network along with the type of traffic. This is especially advantageous when data is of a binary nature, such as images, video, PDF and/or word processor documents, and audio files. This paper describes a prototype of such an application that provides basic stream reconstruction functionality for several TCP/IP protocols. Although still in an early development stage, the application shows great promise.

1 Introduction

Network security usually focuses on detecting intrusion and providing integrity and privacy in networks. These concerns help to protect a network from other entities and the various entities within your network, but what if you needed to see and understand the traffic traversing your network in real time. There are many tools to help you view your network traffic, but almost all of them assume that the user is only concerned with the packet traffic and not the packet contents.

1.1 Sniffers

The main tool for network security professionals are packet sniffers such as Ethereal[1], tcpdump[2], snoop[3], and Sniffem'[4]. Ethereal & Sniffem' provide a graphical user interface, making them easier to use, but they still only provide useful information for users that are fluent in networking internals and have the ability to decipher what exactly is going on. For example, without the use of at least some limiting rules (such as limiting captured traffic to a certain port or IP address) the output of packet sniffing software is often too voluminous to ascertain what traffic trends are present. A good networking professional will know which rules will be the most effective to ascertain what is causing a certain problem. However, not knowing the proper rules will result in obtaining useless information from the network.

1.2 Streams

What is needed is the ability to see, in their entirety, any files being transferred or other transactions that are occurring on the network. There are network tools available today to do this, such as Iris[5], but such tools are very expensive and often use proprietary output formats. The goal with this project was to produce a prototype application, written in Perl[6], to process the raw output from tcpdump, reconstruct any identifiable streams and perform some basic statistical analysis, and produce output in simple HTML. Each stream identifies a single logical transaction between a client and a server. In the case of an HTTP[7] transaction, a stream is composed from a single request from a client and the response from the server. This is true for many protocols, but for others, such as SMTP[8] and POP[9], the stream is composed from all of the requests from a client and all of the responses from the server. The prototype Packet Stream Reconstructor (PSR) only handles several protocols at the moment, but shows promise in being a very useful network tool.

2 Network Information

In order to make large amounts of output from packet sniffing software understandable, it needs to broken down into its logical components. The first level of this is the packet type. In the PSR program, all packets are categorized into one of three types: ARP, IP, and generic or unknown (IEEE). By examining ARP packets, we will know what kind of

discovery activity is going on in the network. This may not be of great importance, but it might show trends such as a network entity attempting to contact all of the machines in a subnet using non-broadcast or non-multicast protocols. A high presence of ARP packets may also indicate a host attempting to spoof the IP address of another host on the network. Next are IP packets, which will be discussed in great detail below. The last type, IEEE, covers all packet types that are unknown to the program. Usually, these packets are IPX or Appletalk packets. Future versions of the software should be able to analyze these packets as well for trends.

2.1 IP Packets

The next level of detail concerns only IP packets. Here the focus is on four different types of packets: ICMP (type 1), TCP (type 6), UDP (type 11), and unknown (any other type). When testing the PSR program, 99.6% of the average IP traffic fell into one of the three known categories. High levels of ICMP traffic may indicate a ping flood attack, either direct or reflexive. Steady ICMP traffic from one host to another may indicate either an errant 'ping' that was left running or the possibility of an open itunnel or icmptunnel[10] on the network, which allow for hackers to tunnel telnetd or some other network service through ICMP packets. The next type of IP traffic, TCP, will be discussed in detail below. The last type of traffic, UDP, in the PSR test runs, was typically for DNS resolutions (port 53). However, odd UDP traffic may indicate a network entity attempting to do a thorough portscan[11] on a network device for an open tftp port, a favorite back door for hackers, or some other open service.

2.2 TCP Packets

The last level of detail concerns only TCP packets. Most of the IP traffic examined during testing was TCP traffic. Indeed, when most people think of Internet traffic, they are thinking about TCP traffic. All e-mail, web, network filesystem (NFS, CIFS/Samba), file sharing, database and instant messaging traffic is typically TCP traffic. For this prototype, focus was placed on e-mail and web traffic. More specifically, all SMTP (Simple Mail Transport Protocol), POP (Post Office Protocol), IMAP[12] (Internet Message Access Protocol), and HTTP (Hyper-Text Transfer Protocol) traffic packets were examined. Statistics for the packet hosts and counts are generated for these protocols, just like the traffic at the other levels, but for these specific protocols 'streams' of traffic were identified and reconstructed into their original format.

3 Packet Structure

In order to process a packet stream, an application must first identify the different types of packets that are traversing the pipe from the packet sniffer to the stream analyzer. As the packets are piped to the PSR application, their type is identified and recorded. If the packet is an IP packet, it is sent to an instance of an 'IP' object that further analyzes it and, if applicable, sends the packet on to an object that performs stream reconstruction.

Although the PSR application does not open a direct connection to the network interface controller, it still needs a rather low level of knowledge of packet structure in order to correctly identify and process the packets. Table 1 above shows a sample breakdown of a packet. This particular example is a packet from an HTTP response, although the payload isn't shown. The PSR application deconstructs each packet into each of the separate fields outlined in the table and the relevant data is stored in a packet data structure. Where needed, the packet bytes were joined and converted from hexadecimal into decimal or a similar format that was more easily identified and readable.

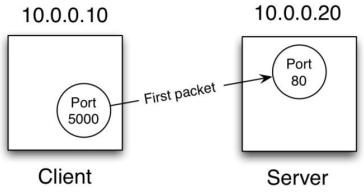
Type	Size (bytes)	HEX	Decimal or examples
Ethernet Header			
Destination MAC address	6	00 04 9a 86 fc 17	
Source MAC address	6	00 b0 d0 20 a4 96	
Packet Type	2	08 00	8 => IP
IP Header			
Version & Length	1	45	4 & 20
Services	1	00	
Total Length	2	00 40	64
ID Number	2	5b 2f	23343
Flags & Fragment offset	1	40 00	
Time to live (TTL)	1	40	64
Protocol	1	6	1 => ICMP, 6 => TCP, 11 => UDP
Header checksum	2	3e 57	
Source IP address	4	41 65 66 39	65.101.102.57
Destination IP address	4	86 81 73 12	134.129.115.18
TCP Header			
Source Port	2	9c 71	40049
Destination Port	2	00 50	80
Sequence number	4	c4 14 dc 5f	3289701471
Header length	1	b0	44
Flags	1	2	02 => SYN
Window Size	2	80 52	32850
Checksum	2	2a 13	
Options	var.		

***Table 1**: Breakdown of a typical TCP/IP packet*

Although this task was tedious, a great deal of knowledge was gained about the structure of packets and how this structure changes during and between connections. However, it became apparent that in order to have an application that can identify and process all packets and packet streams, the application would have to have a working knowledge of all packet types and Internet protocols! In effect, PSR would need an application level IP stack capable of processing multiple types of Ethernet traffic along with the definitions of hundreds of protocols and know the correct way of reconstructing streams for all them. It is hoped that there will be some interest in the computing community in this application and that others will help to expand and enhance the functionality of PSR.

4 Stream Analysis

In order to reconstruct a packet stream, one must recognize that any stream is comprised of packets sent from a specific port on one IP address to another specific port on another IP address. By looking at the first packet in such a stream, the service that is being contacted can be identified and the IP addresses determine the direction of the stream. For example, host 10.0.0.10 uses port 5000 to contact port 80 on 10.0.0.20. We can identify this as a web stream, since the second host was contacted on port 80. The next packet we see from either 10.0.0.20 and using port 80 or 10.0.0.10 using port 5000 can be considered part of the same stream. In this case, 10.0.0.10 is referred to as the 'client', as it initiated the connection and host 10.0.0.20 is the 'server', as it had a service listened on port 80 for incoming connections (see Illustration 1 below). Each stream is identified by its 'stream ID', which is a concatenation of the IP addresses involved (server, then client), as well as the client port, and if necessary, the IP ID Number (see Table 1 above).



Stream ID: 10.0.0.20-10.0.0.10-5000

Illustration 1: Stream ID / Client-Server

4.1 Decoding Transactions

A 'stream' constitutes a single connection and transaction. The transaction may contain several commands from the client to the server, such as the case with SMTP and POP traffic. However, for the most part, is it usually a single command from the client to the server and the response from the server. Currently, what constitutes a stream for a particular protocol is coded into the class for that particular protocol handler. In the future, this will hopefully be an option that is selectable when the output is viewed, rather than when packets are processed, as it is now. Table 2 (below) shows a sample POP

Sample POP transaction between a client & server	
Client	**Server**
--> Connect	
	+OK ready
USER username	
	+OK Password required for username.
PASS password	
	+OK username has 1 visible message
RETR 1	
	+OK xxx octets (message follows)
QUIT	

Table 2: Sample POP transaction - a single 'stream'

transaction between a client and server. The client first makes a connection, then provides their user identifier followed by their password. The server responds that a single message is available, which the client then requests. After the server has sent the message to the client, the client requests an end to the session at which point the server sends a 'goodbye' message and closes the connection. Although there is interaction between the client and the server, this is considered a single transaction and is decoded as such by the PSR application.

4.2 HTTP Streams

Decoding streams of IMAP and SMTP traffic is nearly identical to decoding streams of POP traffic. However, HTTP traffic proved to be much more complex. First an HTTP client can request a 'Keep-Alive' connection with the server. This means that unlike usual HTTP traffic where a client makes a single request and receives a single reply, multiple requests and multiple responses are sent along the same port connection. In the PSR application, these connections should not be considered a single stream, since each HTTP request should be considered to be a different transaction. Luckily, these separate requests are identifiable by examining the IP ID Number (see Table 1 above) and concatenating this number to the stream ID. Another problem with the HTTP protocol is the fact that a server may respond with a header indicating a 'Transfer-encoding' type of 'chunky'. This means that the HTTP server breaks the response into several variable-length 'chunks' before sending the response to the client. To add to the complexity, these

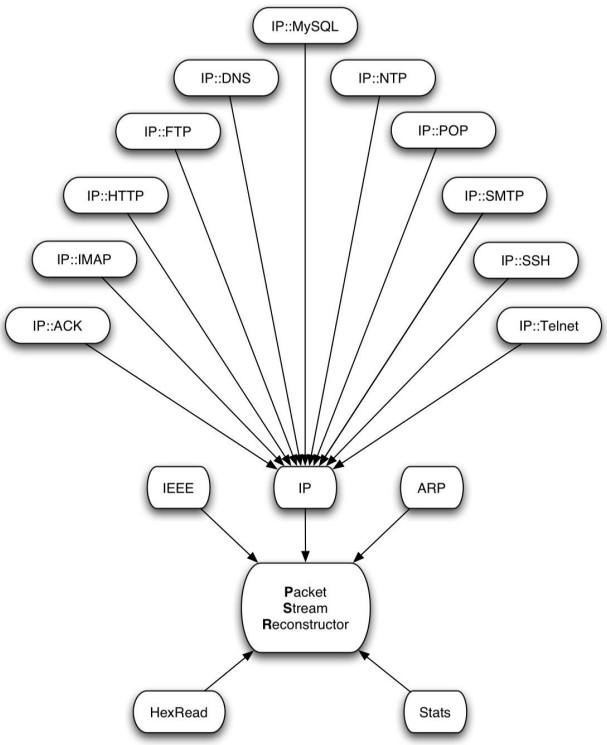


Illustration 2: Simplified class diagram for PSR system

chunks can span several packets. To make the application able to process these odd types of streams, as well as make it extendable to other similarly complex protocols in the future, an object-oriented code development model was used, which allowed all such oddities to be coded into a single class. Thus, the IP object doesn't need to know that such complex streams are being seen on the network, only the HTTP object sees and decodes these streams. As can be seen Illustration 2 above, any protocol extensions can be easily added onto existing system. Each class needs only to define three methods: new() (the constructor), add_packet() and output_stream(). add_packet() receives a packet and determines if it is a stream or not and stores the packet properly and output_stream() simply provides the proper HTML output for that particular packet type.

5 Reporting

In any application that takes some type of input, it is assumed that some sort of output will be generated. The PSR application is no different and generates multiple HTML pages in a predefined directory. The main page shows statistics on the total number of packets, the number of ARP, IEEE (generic/unknown), and IP packets, and finally a list of a links, one for each stream that was found. Each of the protocol count labels is a clickable link that will show additional information, such as MAC & IP addresses and any ports involved in the connections sorted by either MAC or IP address on one side and on the other side sorted by number of packets seen or by port/service number (see Text 1 below). Although this is a rather simplistic report, it can be very useful. As described earlier, certain simple trends can indicate various problems, and these trends are easily seen in these reports. For example, if the IP report shows many packets from many sequential ports on a single host, this would indicate that the host is connecting (or attempting to connect) to either multiple hosts or repeatedly to a single host. This would indicate either a port scan or possibly an attack. Another example would be if many packets are seen involving port 3306, the MySQL daemon port, and such a server is not supposed to be present in the current

PSR: IP Stats - Fri Mar 11 14:58:33 2005

---

19 IP packets were seen.

19 TCP/IP requests were seen.

Source IP addresses: 65.101.102.57 -> 10 134.129.115.18 -> 9	Destination IP addresses: 65.101.102.57 -> 9 134.129.115.18 -> 10
Source ports: 80 -> 10 58698 -> 9	Destination ports: 80 -> 9 58698 -> 10

Source IP addresses: 65.101.102.57 -> 10 134.129.115.18 -> 9	Destination IP addresses: 134.129.115.18 -> 10 65.101.102.57 -> 9
Source ports: 80 -> 10 58698 -> 9	Destination ports: 58698 -> 10 80 -> 9

Text 1: Sample IP Statistics

network. When looking at the output from a packet sniffer, such traffic might have been noticed, if one was looking for it, but it might have been overlooked.

Lastly, of great interest to most, will be the stream reconstructions, which are also on the main HTML page (see Text 2 below). Beside each stream identifier is the type of TCP packet stream that was reconstructed. In the prototype, this is HTTP, IMAP, POP, or SMTP. Clicking on the stream identifier will bring up the human visible reconstruction of the stream. In the case of an HTTP stream, the HTML, graphic, or file type that was transferred will be displayed. In the case of a POP or IMAP stream, the client commands and server responses, as well as any messages will be displayed. Also as a part of the POP stream report, any USER and PASS commands (showing the username & password for the connection - see Table 2 above on page 4) are highlighted in red and the IMAP report shows the 'login' command similarly highlighted. Finally, in the case of an SMTP stream, the client commands and server responses are again show, this time with the "mail from:" and "rcpt to:" commands highlighted in red, allowing the security agent to easily see who the mail was from and who its intended recipient is.

PSR: Sun Dec 12 16:23:23 2004

4682 packets processed
Arp: 206
IEEE: 363
IP: 4113

---

7 IP streams processed
S: 134.129.115.4-66.44.132.62-33203-25c5 - HTTP - html
(Header)
S: 134.129.115.4-66.44.132.62-33203-2581 - HTTP - html
(Header)
S: 65.101.102.57-65.101.102.59-1046 - POP
S: 134.129.115.4-64.124.85.91-33890-cff9 - HTTP - html
(Header)
S: 134.129.115.4-64.124.85.91-33890-d015 - HTTP - html
(Header)
S: 134.129.115.4-66.44.132.62-33204-257f - HTTP - jpeg
(Header)
S: 134.129.115.4-68.77.240.28-52075 - HTTP - html
(Header)

Text 2: Sample output of main HTML page from PSR

Also of note is that under an HTTP stream there is a link labeled "Header" that will show the headers of the GET request and HTTP response (see Text 3 below). In the future, it would be possible to use the information from these headers to modify the HTML tags in reconstructed HTTP streams to allow for images, applets, and included Javascript code to all properly show up and function in such reconstructions. Currently, none of the HTML

code is modified. The header information would also make it possible to recreate a partial mirror of any web sites whose pages we have seen. It would also be possible to modify the stream reconstruction code to **not** reconstruct any streams for files that currently exist on the mirrored filesystem and have not been modified since they were placed there. Other protocols would have a similar format in the main HTML page. For example, an FTP transaction would have a file associated with it that would be stored separately from the request for the file. Also, a MySQL result would be displayed in an HTML table in a separate file from the SQL request that generated the result set.

```
GET /~nem/ HTTP/1.1
TE: deflate,gzip;q=0.3
Connection: TE, close
Host: empyrean.lib.ndsu.nodak.edu
User-Agent: lwp-request/2.06

HTTP/1.1 200 OK
Date: Mon, 06 Sep 2004 04:12:56 GMT
Server: Apache/2.0.48 (Unix) mod_ssl/2.0.48 OpenSSL/0.9.7c
Accept-Ranges: bytes
Connection: close
Transfer-Encoding: chunked
Content-Type: text/html; charset=ISO-8859-1
```

Text 3: Sample output of HTTP header reconstruction

6 Conclusion

To take this project from a prototype, written in Perl, to an application that would be written in C or Java, would take a considerable amount of time and effort. Especially when the desired outcome is to have an application that processes packets of many different types and protocols. Many obstacles were encountered in reading the output from tcpdump correctly and reconstructing the streams found within. From the problem of the varying length of the HTTP protocol, to just ignoring the generic or unknown packets proved to be time consuming tasks in the development of the application. Many such tasks would be needed to be overcome in order to have a flexible tool that works in many different network environments. The PSR application, even in this early stage, is a useful tool that will hopefully, someday, become an essential addition to every network administrators set of applications.

References:

[1] Ethereal - http://www.ethereal.com/

[2] tcpdump - http://www.tcpdump.org/

[3] snoop - Part of the Solaris operating system - http://wwws.sun.com/software/solaris/

[4] Sniffem' - http://www.sniff-em.com/

[5] Iris Network Traffic Analyzer - http://www.eeye.com/html/products/iris/

[6] Perl - http://www.perl.com/

[7] HTTP (Hyper-Text Transfer Protocol) RFC - http://www.w3.org/Protocols/rfc2616/rfc2616.html

[8] SMTP (Simple Mail Transport Protocol) RFC - http://www.ietf.org/rfc/rfc821.txt

[9] POP (Post Office Protocol) RFC - http://www.ietf.org/rfc/rfc1939.txt

[10]icmptunnel - http://www.detached.net/icmptunnel/index.html

[11]Portscanning - see nmap: http://www.insecure.org/nmap/

[12]IMAP (Internet Message Access Protocol) - http://www.imap.org/