A SIMPLE WAY TO CAPTURE NETWORK TRAFFIC: THE WINDOWS PACKET CAPTURE (WINPCAP) ARCHITECTURE

Mihai Dorobanțu, M.Sc., **Mihai L. Mocanu,** Ph.D.

Department of Software Engineering, School of Automation, Computers and Electronics, University of Craiova, 13 A.I. Cuza Str., 1100 Craiova, Romania
Phone: (251) 435724/ext.163, Fax: (251) 438198, *Elm: dmihai@nt.comp-craiova.ro*

Abstract: The paper describes the software architecture for packet capture and network analysis for Win32 platforms. Library calls from a component added to the OS, called ***WinPCap***, can be used to capture network packets and to generate traffic. Traffic generation can be then used to test software components (capture applications) or hardware components (network adapters, modems) in a large variety of tools (applications) specifically designed for network analysis, troubleshooting, security and/ or monitoring. A tool called PCapture, designed and implemented by us, is presented in the final part of the paper.

Key words: packet capture, network analysis

Introduction

The software architecture for packet capture and network analysis for Win32 platforms we intend to describe in this paper contains a library (***WinPCap***), which adds to the operating system the ability to capture efficiently the traffic from different kinds of networks using the network adapter of the machine.

The systems based on Unix platform contain a kernel component used as a capture driver named ***BPF*** (Berkeley Packet Filter). This part of the kernel is the base of the ***libpcap*** library. Unlike Unix platforms, Windows operating systems haven't such a component.

The implementation of a similar library used to capture data is more difficult to realize because it calls for an interaction with the networking part of the kernel. This is more difficult in the "Microsoft world" where kernel source code is not available.

The WinPCap Architecture

WinPcap is an architecture for packet capture and network analysis for the Win32 platforms. It includes a kernel-level packet filter, a low-level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll, based on libpcap version 0.6.2).

The packet filter is a device driver that adds to Windows 95, 98, ME, NT, 2000 and XP the ability to capture and send raw data from a network card, with the possibility to filter and store in a buffer the captured packets.

WinPCap can be used to capture the network packets and to generate traffic (sending packets through the network).

Packet.dll is an API that can be used to directly access the functions of the packet driver, offering a programming interface independent from Windows.

Wpcap.dll exports a set of high level capture primitives that are compatible with libpcap, the well-known Unix capture library. These functions allow to capture packets in a way independent from the underlying network hardware and operating system.

WinPCap can capture data sent through current machine (which is used by WinPCap) or to any computer from the LAN (only if a hub or a switch does not direct the traffic - these kinds of equipments tend the packets through the destination computer). Generally, generating the traffic is used to test some software components (capture applications) or hardware components (network adapters, modems).

Under these considerations, WinPcap can be used by different kinds of tools for network analysis (for troubleshooting, security and monitoring):

- Network and protocol analyzers
- Network monitors
- Traffic loggers
- Traffic generators
- Network intrusion detection systems (NIDS)
- Network scanners
- Security tools

WinPCap Structure

To capture data from the network, a capture application must directly interact with the network adapter. Therefore the operating system must offer a set of primitives for capture with a view to communicate with the adapter. The purpose of these primitives is to capture network packets transparently from the point of view of the user and to transfer them to the calling application.

This part of the kernel should be quick and efficient in order to capture all the packets in real time without losing information.

The ***WinPCap*** Architecture is a hierarchical structure on 3 levels (from the network adapter to an application), shown in figure 1.

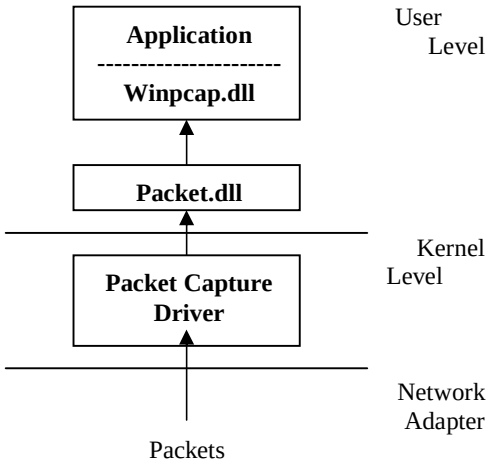

Figure 1. The WinPCap Architecture

At the lowest level there is the network adapter. It is used to capture the packets that circulate in the network. It can be set to "promiscuous mode" that means it will accept all the packets even if they are not intended for current computer.

The ***Packet Capture Driver*** is the lowest software level of the capture structure. It is a part of the kernel and interacts with the network adapter to obtain captured packets. It supplies the application a set of functions used to read /write data from the network at data-link level.

Packet.dll works at user level, but it is detached from the capture application. It is a dynamic link library that separates the application from capture driver providing a system-independent capture interface. It allows user's application to be executed on different Windows operating systems without being recompiled. It represents a set of API functions used to access the capture driver directly.

WinPCap.dll (the third level) is a *static* library that is used by the packet captures part of the application. It uses the services exported by Packet.dll, and provides the applications a higher level and a powerful capture interface. It is statically linked; it means it is part of the application that uses it.

The user interface is the highest part of the capture application. It manages the interaction with the user and displays the result of a capture.

The user's application receives the captured information and interprets it in order to obtain significant data. Because of the great number of communication protocols, the user's applications should be flexible and modular.

A final remark to this description should be useful: the use of term *packet* here may be not very accurate but is most expressive. Capture process is done at Data-Link Layer and the PDU (Protocol Data Units) for this layer is the frame (according to ISO/OSI Model).

The Format of a WinPCap Application

In WinPcap 3.0, the final stable release of this package, the main improvements worth to be mentioned are:

- kernel buffering rewritten from scratch
- experimental support for remote capture.

The support for SMP machines has been included starting from version 3.0, but only physical interfaces are supported (this is a limitation of Windows and not of WinPcap). All WinPcap can run on the main Win32 operating systems: Windows 95,98,ME, NT4 and 2000, but for Windows XP, WinPcap version 2.3 or higher is required.

The WinPCap action is based on packets capture at the network adapter level. Therefore, any application using WinPCap must follow the steps summarized below:

- Specify the network adapter which will be used for capture
- Initialize winpcap (mention the network adapter if the capture is on-line)
- Offer a pattern for the captured data (a TCP/IP structure for example), so it can be done a cast at this pattern and obtain significant information (complex applications offer patterns for majority protocols)
- Analyze captured data according with the application type (captured data can be saved and processed off-line)
- Close the capture session

The first step in using the WinPCap Library is to select a network adapter. This adapter can be specified in two ways. The first way is by adding its name as argument in the command line (this works with a trivial application):

```
int main(int argc, char *argv[])
{
  char *dev = argv[1];
  printf("Network Adapter: %s\n", dev);
  return(0);
}
```

The second technique uses the function ***pcap_lookupdev()*** to extract the list of all the adapters presented on current machine:

```
int main()
{
  char *dev, errbuf[PCAP_ERRBUF_SIZE];
  dev = pcap_lookupdev(errbuf);
  printf("Network Adapters:%s\n", dev);
  return(0);
}
```

After the selection of the network adapter, the next step is to initialize the capture session. This is done with use ***pcap_open_live()***. This WinPCap function opens a capture session. It returns a handler to the current session and it is used for on-line capture. A similar function is used to retrieve saved data- ***pcap_open_offline()***.

```
...
pcap_t *handle;
handle = pcap_open_live(dev, BUFSIZ, 1,
   0, errbuf);
...
```

The **pcap_open_live()** function specifies the network adapter(dev) and sets it to normal/ promiscuous mode (third argument). An important feature of WinPCap Library is that it can filter the

traffic. Many times, we are interested in watching only one kind of traffic (for example data sent/received on 23 port - telnet). In these cases filters are set to obtain only relevant information.

To filter the traffic, WinPCap Library offers two functions: ***pcap_compile()*** and ***pcap_setfilter()***. That is, first time a filter must be compiled (using the first function) and then applied. Only after the capture session is opened, a filter can be applied.

#include “pcap.h”

```
...
// Handler to current capture session
   pcap_t *handle;
// Network Adapter
   char dev[] =
"eth0";
// Error Buffer
   char errbuf[PCAP_ERRBUF_SIZE];
// Filter Variable
struct bpf_program filter;
// Filter Expression
char filter_app[] = "port 23";
// Network Adapter Mask
bpf_u_int32 mask;
// Network Adapter IP Address
bpf_u_int32 net;
// Obtain IP Address and Mask
pcap_lookupnet(dev, &net, &mask, errbuf);
// Open Capture Session
handle = pcap_open_live(dev, BUFSIZ, 1, 0,
errbuf);
// Compile the filter
pcap_compile(handle, &filter, filter_app, 0,
net);
// Apply compiled filter
pcap_setfilter(handle, &filter);
…
```

The WinPCap Library contains the function ***pcap_lookupnet()*** which retrieves IP address and mask of a specified network adapter. The code above first opens a capture session and watches the 23-port traffic by applying a simple filter.

The most important thing is to capture the packets (so far, we open a capture session and apply a filter without capturing anything). There are two techniques for capture data. The first technique uses the function ***pcap_next()*** to capture only one packet:

```
int main()
{
// Handler to current capture session
   pcap_t *handle;
// Network Adapter
   char dev[] = "eth0";
// Error Buffer
   char errbuf[PCAP_ERRBUF_SIZE];
// Filter Variable
   struct bpf_program filter;
// Filter Expression
   char filter_app[] = "port 23";
// Network Adapter Mask
   bpf_u_int32 mask;
// Network Adapter IP Address
   bpf_u_int32 net;
// Packet Header
   struct pcap_pkthdr header;
// Captured Packet
   const u_char *packet;
// Obtain Network Adapter
   dev = pcap_lookupdev(errbuf);
// Obtain IP Address and Mask
   pcap_lookupnet(dev,&net,&mask, errbuf);
// Open capture session in promiscuous mode
   handle = pcap_open_live(dev, BUFSIZ, 1, 0,
   errbuf);
// Compile & Apply the Filter
    pcap_compile(handle, &filter, filter_app,
    0, net);
    pcap_setfilter(handle, &filter);
// Capture one packet
   packet = pcap_next(handle, &header);
// Display Packet Length
   printf("Captured a packet with length of
   [%d]\n", header.len);
   // Close Capture Session
   pcap_close(handle);
   return(0);
}
```

The second technique for capturing a packet is more complicated but more helpful. It uses a loop to capture all packets received this time. Once the adapter is opened, the capture can be started with ***pcap_dispatch()*** or ***pcap_loop()***. These functions are similar, the difference is that ***pcap_dispatch()*** is granted to return when it expires while ***pcap_loop()*** doesn't return until the specified number of packets have been captured, so it can block for an arbitrary period on a few utilized network. The call **pcap_loop()** is used in simple applications, while ***pcap_dispatch()*** is normally preferred in a more complex program.

After we capture one packet (or more), we may want to interpret it. The natural way to interpret a packet is to cast it to a defined pattern. Complex applications define patterns for different kinds of communication protocols and display information on any captured data.

Let’s consider we grabbed a TCP/IP packet and we want to extract the information on this type of protocol. Because it is a TCP/IP packet, it should have an Ethernet Header (Ethernet Network), an IP Header and a TCP Header followed by sending data. First step in interpreting a packet is to cast it to Ethernet pattern. Now we are obtaining information about next protocol (IP Protocol for our example). The packet without Ethernet Header is cast again to IP pattern and analyzed. If we have a TCP/IP packet, the process ends by converting it to a TCP pattern. Now we have all the information about this captured packet.

```
…
// Ethernet Header
   const struct hdr_ethernet *ethernet;
// IP Header
   const struct hdr_ip *ip;
// TCP Header
   const struct hdr_tcp *tcp;
// Packet Data
   const char *data;
// Headers Length
   int size_ethernet = sizeof(struct
hdr_ethernet);
   int size_ip = sizeof(struct hdr_ip);
   int size_tcp = sizeof(struct hdr_tcp);
// Simple Conversion
  ethernet=(structhdr_ethernet*)(packet);
  ip=(structhdr_ip*)(packet+size_ethernet);
  tcp = (struct hdr_tcp*) (packet+size_ethernet
      +size_ip);
  data = (u_char *)(packet+size_ethernet+
      size_ip+size_tcp);
```

As an Ethernet Header has 14 bytes, IP Header 20 bytes and TCP Header 20 bytes, the **u_char** pointer represents the memory address where we can find information about the captured packet, and header

dimensions can differ from a platform to another. This is the reason why we were using the sizeof() function.

The WinPCap Architecture is generally used for capture traffic but it is capable to generate traffic by sending packets through the network. Generating traffic is useful in analyzing capture applications and hardware devices. We are interested in the percentage of lost packetsfor instance, an application that has captured only 100 packets from 1000, it's obvious that it should be retouched.

The function used for sending packets is ***pcap_sendpacket()***. It sends packets formatted by us (according to communication protocols) through specified computer.

Since WinPcap is implemented as a protocol, therefore it is able to capture the packets, but it can't be used to drop them. The filtering capabilities of WinPcap work only on the sniffed packets. In order to intercept the packets before the TCP/IP stack, an intermediate driver needs to be created by or provided to the network administration team.

The PCapture Application

PCapture is an interactive tool which uses the WinPCap architecture and C/C++ calls for packet captures. Although we used for its interface the MFC (Microsoft Foundation Class) functions, PCapture has a certain degree of independence from the OS, by using the "compatible" calls of WinPCap. Use of threads reduces somehow the predictibility of application behavior only to systems similar to Windows NT, 2000 or XP (we noticed non-deterministic behavior during tests on Windows 98 or ME).

The tool is just a preliminary step; it achieves the capture of all packets in teh network and the specification of their type. For "standard" packet types like TCP/IP and UDP the component fields of the headers are specified, shaped as a tree structure. Going towards the superior level (Application) requires the definition of patterns, corresponding to the types of protocols on this level.

This tool has options for loading and saving capture sessions, using disk files. It is also very simple and friendly from the UI (user interface) point of view. The user can easily select the capture mode, filters, adaptors a.o. An interesting and useful feature proved to be the refreshing option, by which re-filtering of already captured packets is achieved. In other words, although the capture session may be ended, if the number of packets is too big making their analysis too difficult, a new filter can be selected and the screen refreshed accordingly. A snapshot of the captures obtained during a session is shown in figure 2.

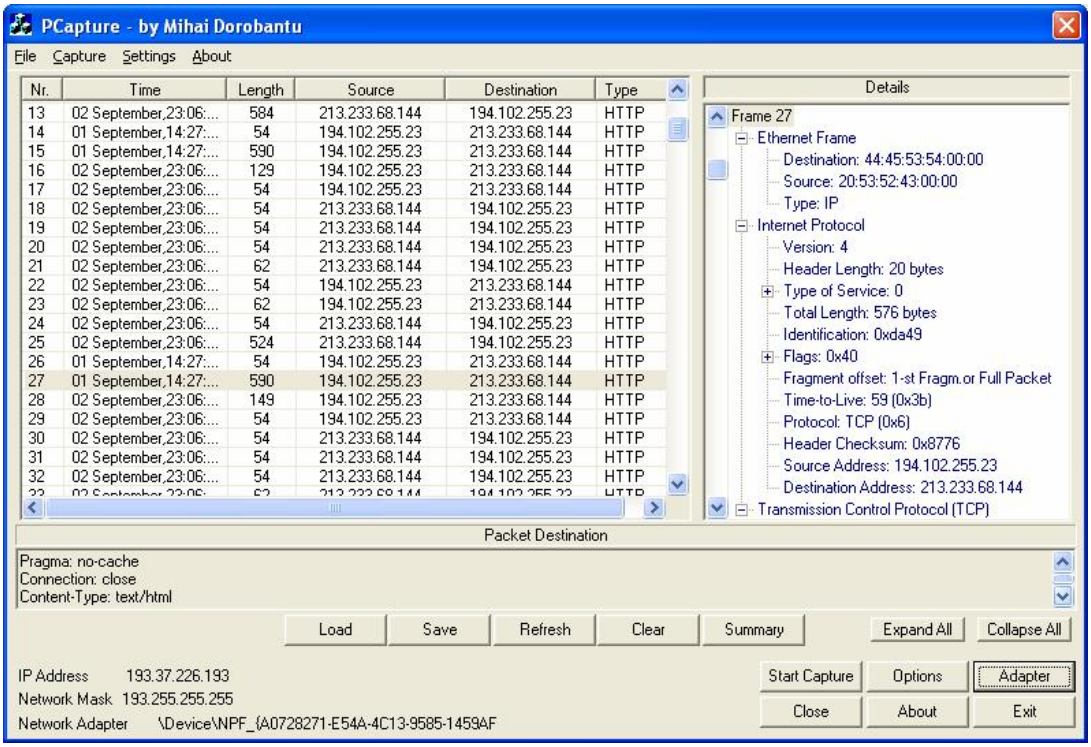

Figure 2. The PCapture Tool

Conclusion

The purpose of this article is to present a simple way of capturing the network traffic using the functions supplied by the WinPCap Architecture. This library contains a lot of useful primitives and it is compatible with another library used in the same purposes **libpcap** library.

References

WinPCap Home Page – http://winpcap.polito.it

Loris Degioanni, Development of an Architecture for Packet Capture and Network Traffic Analysis, Graduation Thesis, Politecnico Di Torino (Turin, Italy, Mar. 2000), http://winpcap.polito.it/docs/th_degio.zip

Fulvio Risso, Loris Degioanni, An Architecture for High Performance Network Analysis, *Proceedings of the 6th IEEE Symposium on Computers and Communications (ISCC 2001)*, Hammamet, Tunisia, July 2001

Tim Carstens, Programming with pcap, tutorial, http://broker.dhs.org/pcap.org

Martin Casado, Packet Capture With libpcap and other Low Level Network Tricks, tutorial, http://www.cet.nau.edu/~mc8/Socket/Tutorials/section1.html