Packet Pal
A Network Packet Analysis and Penetration Testing Utility

Author:
Kurtis Kopf

Advisor:
Dr. Hal Berghel

In partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

School of Computer Science
Howard R. Hughes College of Engineering
University of Nevada, Las Vegas

Fall, 2007

Abstract

Packet Pal: A Network Packet Analysis and Penetration Testing Utility

by
Kurtis Kopf

Dr. Hal Berghel, Examiniation Committee Chair
Professor of Computer Science
University of Nevada, Las Vegas

TCP/IP network packet crafters have been in circulation for twenty years. Products such as Ethereal, Hunt, and Achilles have been widely used by network “hackers” for over a decade. Of course, these tools are also useful to network analysts and network forensics professionals for packet analysis and penetration testing. However, the disadvantage of using “hacker tools” for legitimate analysis and testing purposes is that the crafted packets may become “live” on the network. For most purposes, especially in the educational, classroom, or laboratory environment, launching live crafted packets is an exceedingly bad idea.

Packet Pal is a network packet crafter that uses the standard libpcap and WinPcap packet-capture libraries while restricting the network behavior to fall within the scope of an organization's security policies. This enables the experimenter or instructor to utilize or teach packet crafting without posing a threat to the host network, network appliances, or computer systems. It also enables the user to launch crafted packet trains toward network appliances within a tightly controlled environment for penetration testing. The user may test firewalls, routers, gateways, wireless access points, etc. systematically and thoroughly for vulnerabilities, without the fear of compromising the host network.

In order to accomplish this goal, Packet Pal uses a convenient GUI to enable the user to craft packets and packet “trains” visually rather than through a command line or text editor. Packet Pal also supports the editing of captured packets. It can operate on a loopback interface to allow for network analysis to be conducted without any network traffic leaving the host machine. An API to allow developers to add editors for new types of packets, allows Packet Pal to be extensible as new network protocols are developed. A plug-in API has also been developed and included to allow further functionality to be added to Packet Pal.

Review Committee

Dr. Hal Berghel, Committee Chairman
Dr. Jan Pedersen, Reader
Dr. Yoohwan Kim, Reader
Dr. Ju-Yeon Jo, Graduate College Representative

Table of Contents

Table of Figures

List of Tables

Introduction

Project Derivation and Goals

Hal Berghel developed the idea for Packet Pal in the late 1990s while teaching network security. At the time, all available packet crafting tools were "invasive," such as **Hunt**[1], **Achilles**, and **Cheops**. According to Berghel, invasive packet crafting is a high-risk activity for two reasons: packet crafting by its very nature is experimental; and the consequences of most interesting modifications are potentially damaging to live networks. What was needed was the functionality of the invasive packet crafters without the risk. Thus, the idea for "non-invasive" or "sterile" packet crafting was born.

The intention was to use the industry standard libpcap library for packet inspection with a **GUI** that resembled **Ethereal** and Wireshark. Van Jacobson, Craig Leres, and Steven McCanne originally developed libpcap while working in the Lawrence Berkeley Laboratory Network Research Group [6]. The libpcap library is widely used for capturing network traffic. WinPcap, an indirect port of libpcap to the Windows platform, was originally developed by Piero Viano at the Politecnico di Torino as his graduation thesis and is currently maintained by Loris Degioanni, Gianluca Varenni, Fulvio Risso, and John Bruno [20]. Gerald Combs originally wrote **Ethereal**. The name was changed to Wireshark in 2006 when Combs left his previous employer. The Wireshark project now has over 500 contributors while Combs maintains the overall code and releases new versions. Wireshark is a powerful network protocol analyzer that provides an in-depth **GUI** for packet analysis [2].

For packet inspection, the **Ethereal** / Wireshark **GUI** had stood the test of time for over a decade. The value added by Packet Pal would be the addition of an editor that could (1) allow the simple editing of a packet, (2) allow the creation of packets to show how multi-packet exploits would appear, and (3) create a simulated environment to show the effect of deploying (1) or (2) against a hypothetical network appliance like a certain model of Cisco Router running a particular version of IOS. The purpose of this product is to implement (1) and (2).

Packet Pal is envisioned as a cornerstone of the Packet Pal Suite, the Packet Pal Primer of which has been online for nearly ten years [1]. The current version of Packet Pal will be made available as an open-source download on the Packet Pal website in the near future.

The goals of this product are as follows:
- Enable a user to graphically capture network communications
- Enable a user to graphically modify network communications

[1] Glossary terms in bold are defined in *Appendix C: Glossary*.

- Enable a user to safely re-broadcast modified communications
- Provide an **API** for new communication protocols to be added
- Provide an **API** for various plug-in components to be added
- Provide a familiar program layout for network professionals
- Release Packet Pal as an open-source product to the community

Need for a Graphical Packet Modification Tool

Networking is a critical area of computer science, and as such, is a common subject in universities worldwide. Switched phone lines, wired networks, wireless 802.11 networks, mobile phone networks, Bluetooth, and others are in widespread use around the globe, with more new technologies being designed every day. With these networking technologies comes a plethora of ever-expanding communications protocols specifications and implementations.

In order to maintain pace with the field, a packet analysis or modification tool must have an extensible architecture. Available commercial products, such as NetScan Tools Pro and NS Auditor, lack public **APIs**, which would allow third parties to create extensions for their tools [15] [17]. This leaves the end user dependant upon the priorities and time constraints of the commercial development team for the addition of new network protocols and features needed for specific environments. Open-source products, such as Wireshark, are able to keep development lag to a minimum while users develop and contribute extensions as needed for new technologies.

The intuitive display of packet information helps to relay important information quickly to the end user. Some tools, such as Wireshark, have the ability to display packet information as parsed text in a layered format. While this accurately portrays the layers of the packet, sometimes, such as with TCP packets, the order of information does not accurately reflect the bitwise construction of the packet, which can be useful information. Wireshark, as well as other tools like NS Auditor, are able to show packet information in hexadecimal side by side with the parsed text. While this method of display is often helpful, it can be difficult to read if the user does not already understand the structure of the packet being viewed. When both views are used in parallel, most of the information is available, but neither of these two methods follows the stacked byte-word representations popular in networking textbooks such as TCP/IP Illustrated [8].

Packet modification, despite all advances in the field of networking, remains a very difficult process. Packets are complex strings of bits, composed of layers representing encapsulated protocols. Editing a packet requires in-depth knowledge of every protocol represented within that packet and the wide variations that arise within each protocol. Due to the level of knowledge required, many tools will only modify certain types of packets with known protocols, such as the open-source tool tcprewrite [9]. Another option is to only allow generation of specific types of traffic to simulate predefined attacks, which is the methodology that some penetration testing tools like NetScan Tools Pro and **Metasploit** follow [15] [14].

With this understanding of the challenges associated with packet capture and modification, Packet Pal was designed to fill the voids left by popular products in the current market. Packet Pal is an open-source tool designed to capture, modify, and rebroadcast packets, using an intuitive **GUI**, with an extensible program structure.

Intended Audience

This product is intended for use on Microsoft Windows XP systems with .NET 2.0 or higher and Windows Vista 32-bit editions. Either operating system requires WinPcap version 4 or higher to be installed in order to access network hardware. At the time of this presentation, there are known issues, detailed in the *Development Environment* section, with the Windows Vista 64-bit operating system SharpPcap, and WinPcap that prevent Packet Pal from running.

The target users are educators with background knowledge of network communications, students who are learning about network communications or network security, and penetration testers who are testing network systems. The primary purpose of Packet Pal, however, is to be a teaching utility, which weighed heavily in the decision making process during development.

Pre-Development Research

In order to begin development of Packet Pal, it was important to have an understanding of the following background topics:

- The **OSI** and **TCP/IP** networking models
- How operating systems, such as Windows and Linux, process packets
- How **TCP/IP** stacks works
- How individual packet layers are constructed and organized
- The mathematics involved in programming in the networking environment
- How various network protocols can be exploited
- How libpcap and WinPcap interact with the host operating system
- Experience with popular networking tools in order to maintain some parallels with similar tools

Project Background Research

Capturing Packets

There are very few methods of capturing packets at a very low level in any operating system. Different issues arise in each target operating system, as they all implement packet handling and delivery differently. Probably the most widely used method for capturing packets involves programming with the libpcap library, which provides a standard interface in which programmers can interact with an operating system's network implementation [6]. With ports of the libpcap library to nearly every operating system, and several of the most popular networking tools on the market, such as Wireshark, being developed on top of libpcap, libpcap was chosen as a basis for capturing packets [2].

Target Platform

Several rapid prototypes using different platforms, design principals, and tools were developed to gain an understanding of performance and a perspective on the difficulty of constructing Packet Pal in certain environments. One such rapid prototype involved the Java Development Environment and the libpcap interface library jpcap. Although receiving packets was possible, broadcasting packets is not natively supported by jpcap and would require constructing packets with standard java components or another third party library [12].

Windows was originally selected as the target platform for Packet Pal due to its market dominance, but with no official build of libpcap to the Windows environment, the inherent challenges when dealing with the networking layer in order to capture packets directly arise again. To alleviate these issues, the port of libpcap for the Windows environment, WinPcap, is used [20]. The WinPcap library is written as unmanaged C++ code, which means development in Visual Studio requires an in-depth knowledge of the library. In Visual Studio, details such as the organization and public methods of any object in a library compiled in a managed language are available to the developer. These features facilitate the proper use of methods within libraries and reduce development time.

Recently, projects, such as jpcap and SharpPcap, have succeeded in bringing the vast libpcap library into a managed development environment, allowing developers to use the rich features of Visual Studio and create applications more easily. SharpPcap enables C#.NET applications to quickly and easily interact with the WinPcap library. Because C# is a managed language, interacting with any library written in C# within Visual Studio, such as SharpPcap, becomes relatively easy, even without extensive documentation [5].

After successfully developing rapid prototypes in C# with SharpPcap, the original target platform of Windows was attainable, although with some further requirements discussed below.

Development Environment

Windows XP requires .NET 2.0 or higher and WinPcap for Packet Pal to run successfully. The user running Packet Pal in XP must have administrator level access to be able to access the network hardware, otherwise no devices will be found by WinPcap. This same issue exists with all programs that use WinPcap under Windows XP and even libpcap on UNIX environments, where super user access is required [20] [2]. Windows Vista 32-bit variants require only WinPcap to run Packet Pal, since every variant ships with .NET 3.0 or higher. There are, however, some issues with the 64-bit versions of Windows Vista. WinPcap will install in Vista 64-bit versions, but SharpPcap and Packet Pal are unable to communicate with the library. This problem was discovered and diagnosed during the debugging stages of this project. In order to solve this problem, SharpPcap would need to be modified significantly to properly communicate with WinPcap within the 64-bit environment, which is beyond the scope of this project.

Packet Pal was developed in Visual Studio 2005 using the C# language. To develop extensions or modify Packet Pal, Visual Studio 2005 is recommended. WinPcap must be installed on the development machine, and a copy of the SharpPcap library, included with Packet Pal, is also required. Because SharpPcap is an open-source project, it may also be beneficial to have a copy of the source code, which is easily recompiled using Visual Studio 2005.

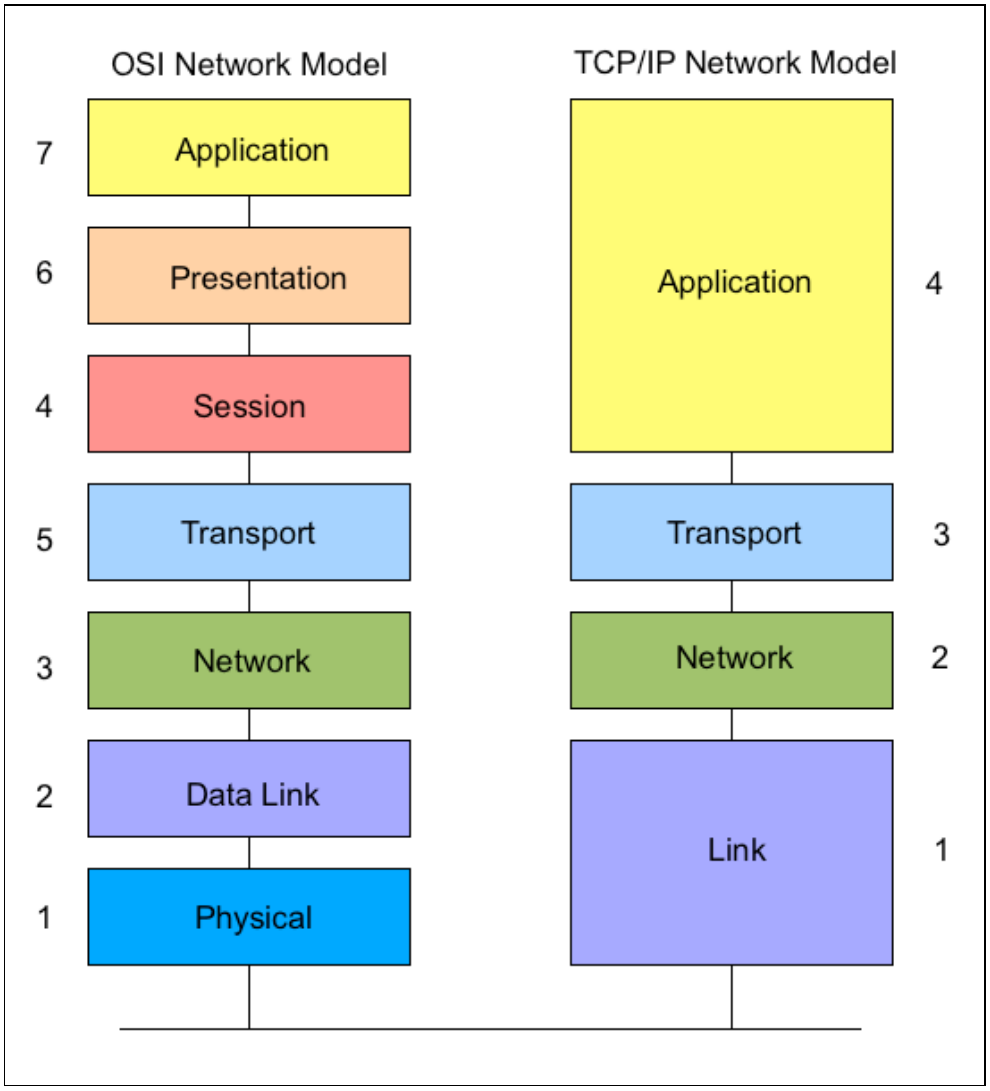
Project Research

Graphical Packet Display

The most challenging part of the design process for Packet Pal involved the **GUI** for the display of individual packet information. The task of developing a coherent interface to deal with raw networking data in an intuitive manner proved very difficult for this project. For any given layer in the **OSI** or **TCP/IP** networking models, there are several protocols that potentially need to be addressed [3] [8]. In addition, each protocol can be fairly complicated, containing numerous fields representing different types of data. There are varying approaches to this problem: layered text, hexadecimal side by side with text, and the stacked byte-word educational model.

The **OSI** and **TCP/IP** layered network models are a good place to begin understanding how packets are constructed, since they are the theoretical basis for most implementations of network stacks [3] [8] [19]. See *Figure 1* for a comparison of the **OSI** and **TCP/IP** models.

Figure 1 - Layered Network Models

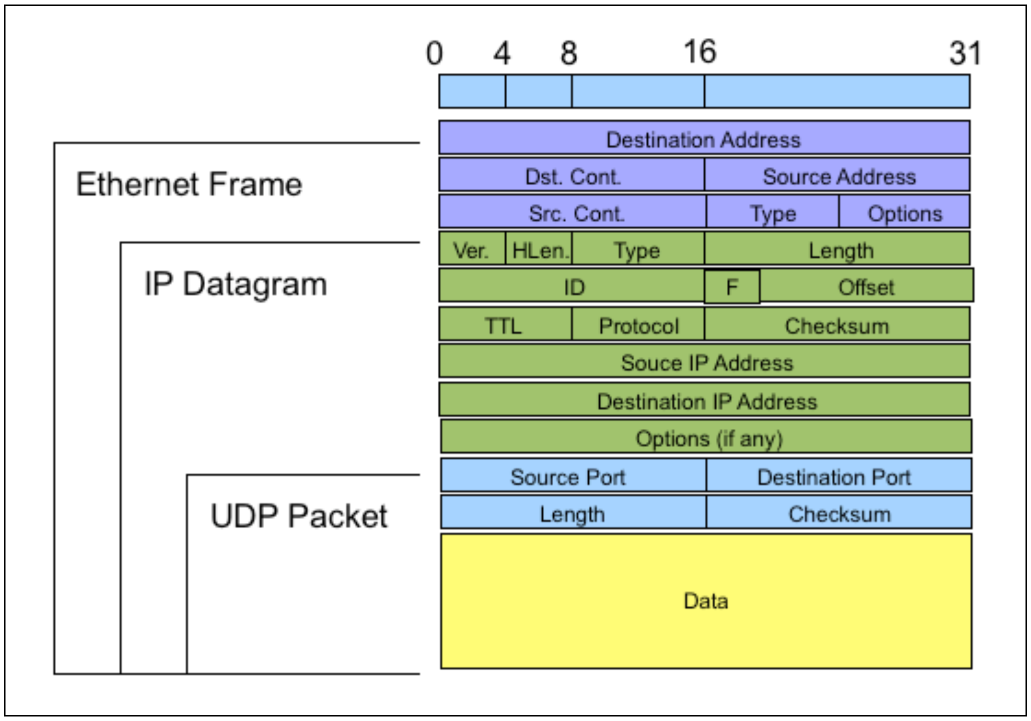


The **OSI** and **TCP/IP** models are essentially the same, constructing network communications based on different tasks being assigned to different levels. These communication tasks are assigned to protocols and programs designed to handle those specific protocols. Packet Pal is based around the **TCP/IP** model because it is more concise and more accurately represents all of the concepts that Packet Pal needs in order to operate. For instance, in a TCP packet, the Ethernet protocol resides within the Link Layer, the IP protocol resides within the Network layer, the TCP protocol resides within the Transport layer, and the original communication by the host program resides within the Application layer. This analogy can be applied to the **OSI** model as well, but the Physical layer represents the network hardware itself, and the host application handles Session, Presentation, and Application. For this analysis, the **TCP/IP** model fits nicely.

In an example network communication, an FTP client and server need to communicate by passing messages and data. The FTP client and server reside at the Application layer on two separate machines. To send a communication, the sending party will create a message based on the FTP protocol and deliver it to the Transport layer. The Transport layer then encapsulates the FTP message within a TCP Packet, and passes the result to the Network layer. The Network Layer then encapsulates the Packet within an IP Datagram and passes the result to the Link Layer. The Link Layer encapsulates the Datagram within an Ethernet Frame and finally broadcasts the packet across the network. Assuming the two hosts are on the same network, and ignoring routing for the time being, the process is completely reversed by the receiving party. Theoretically, each layer needs to know nothing about its surrounding layers, but in practice, small bits of information are passed for efficiency purposes.

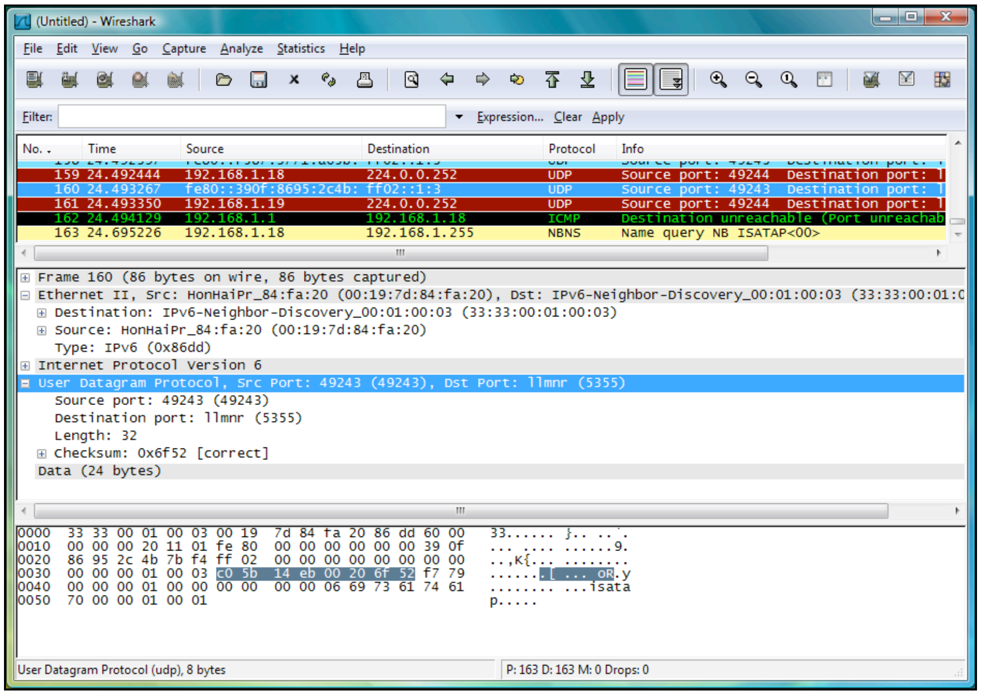
With this understanding of the organization of protocols and how packets are constructed in layers, *Figure 2* shows a stacked byte-word academic representation of a UDP packet. The colors used coincide with the colors in the **TCP/IP** model diagram in *Figure 1*.

Figure 2 - Academic Network Protocol Layout



Wireshark's **GUI** displaying the internals of a UDP packet is shown in *Figure 3*. Wireshark's **GUI** provides a good example showing both the layered text approach (middle frame) and side-by-side hexadecimal and text approach (lower frame).

Figure 3 – Wireshark Displaying a UDP Packet


(Untitled) - Wireshark
File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .  Time       Source                 Destination   Protocol  Info
159 24.492444  192.168.1.18           224.0.0.252   UDP       Source port: 49244  Destination port: l
160 24.493267  fe80::390f:8695:2c4b:  ff02::1:3     UDP       Source port: 49243  Destination port: l
161 24.493350  192.168.1.19           224.0.0.252   UDP       Source port: 49244  Destination port: l
162 24.494129  192.168.1.1            192.168.1.18  ICMP      Destination unreachable (Port unreachab
163 24.695226  192.168.1.18           192.168.1.255 NBNS      Name query NB ISATAP<00>
```

```
⊞ Frame 160 (86 bytes on wire, 86 bytes captured)
⊟ Ethernet II, Src: HonHaiPr_84:fa:20 (00:19:7d:84:fa:20), Dst: IPv6-Neighbor-Discovery_00:01:00:03 (33:33:00:01:0
  ⊞ Destination: IPv6-Neighbor-Discovery_00:01:00:03 (33:33:00:01:00:03)
  ⊞ Source: HonHaiPr_84:fa:20 (00:19:7d:84:fa:20)
    Type: IPv6 (0x86dd)
⊞ Internet Protocol Version 6
⊟ User Datagram Protocol, Src Port: 49243 (49243), Dst Port: llmnr (5355)
    Source port: 49243 (49243)
    Destination port: llmnr (5355)
    Length: 32
  ⊞ Checksum: 0x6f52 [correct]
  Data (24 bytes)
```

```
0000  33 33 00 01 00 03 00 19  7d 84 fa 20 86 dd 60 00   33...... }.. ..`.
0010  00 00 00 20 11 01 fe 80  00 00 00 00 00 00 39 0f   ... .... ......9.
0020  86 95 2c 4b 7b f4 ff 02  00 00 00 00 00 00 00 00   ..,K{... ........
0030  00 00 00 01 00 03 c0 5b  14 eb 00 20 6f 52 f7 79   .......[ ... oR.y
0040  00 00 00 01 00 00 00 00  00 00 06 69 73 61 74 61   ........ ...isata
0050  70 00 00 01 00 01                                  p.....
```

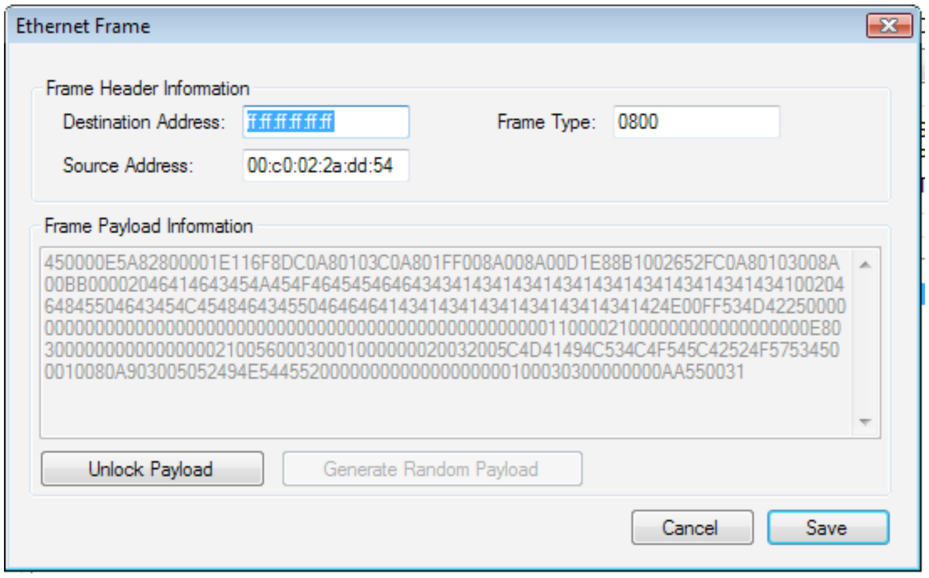
```
User Datagram Protocol (udp), 8 bytes        P: 163 D: 163 M: 0 Drops: 0
```

While Wireshark's **GUI** lends itself well to conveying information about the packet, the interface does not make adding standard data input devices, such as text boxes, a viable option. Data input is necessary for packet modification and should be part of the display interface. Under the assumption that seamless data input is a critical feature, a new model for display based more on the academic model for display was constructed.

While a single packet is constructed of multiple layers, changes made in lower layers of the packet can have a ripple effect throughout the entire packet. For instance, changing the data length in an IP packet can destroy the data of the layer above it, such as UDP or TCP. To compound this problem, if Packet Pal is to maintain the theoretical model for networking, each layer should be parsed as an independent structure. This makes displaying information for all layers within one graphical interface very difficult, as the packets need to be constantly recompiled and reinterpreted as changes are made.

To combat these issues, interfaces for each protocol were designed as separate windows and Packet Pal only allows the editing of one layer of a packet at a time. When changes are made to a packet, it is reparsed from the ground up to determine its structure, accurately reflecting changes that affect other layers. This solution also maintains the educational ideal of separation of the communication protocols. The following figures demonstrate the layers of a UDP packet, as it is displayed in Packet Pal. *Figure 4* shows the Ethernet layer of the packet, *Figure 5* shows the IP layer of the packet, and *Figure 6* shows the UDP layer of the packet.

Figure 4 - Ethernet Layer Display



Ethernet Frame

Frame Header Information

Destination Address: ff:ff:ff:ff:ff:ff

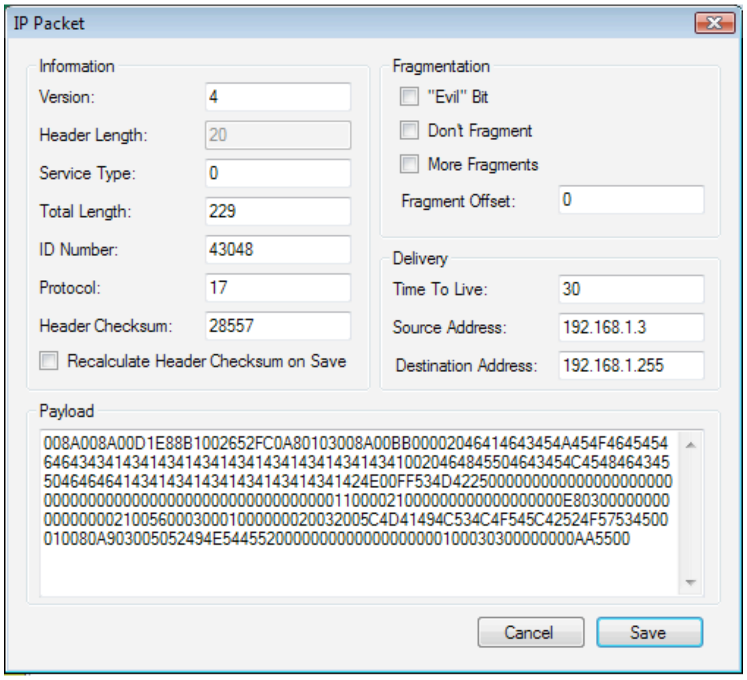
Frame Type: 0800

Source Address: 00:c0:02:2a:dd:54

Frame Payload Infomation

```
450000E5A8280001E116F8DC0A80103C0A801FF008A008A00D1E88B1002652FC0A80103008A
00BB0000204641464345 4A454F4645454646434341434143414341434143414341434143410020 4
648455046434 54C45484643455046464641434143414341434143414341424E00FF534D42250000
00000000000000000000000000000000000000000000000110000210000000000000000000E80
300000000000000002100560003000100000000200320 05C4D41494C534C4F545C42524F5753450
0010080A903005052494E54455200000000000000000000010003030000000 00AA550031
```

Figure 5 - IP Layer Display



IP Packet

Information

Version: 4

Header Length: 20

Service Type: 0

Total Length: 229

ID Number: 43048

Protocol: 17

Header Checksum: 28557

☐ Recalculate Header Checksum on Save

Fragmentation

☐ "Evil" Bit

☐ Don't Fragment

☐ More Fragments

Fragment Offset: 0

Delivery

Time To Live: 30

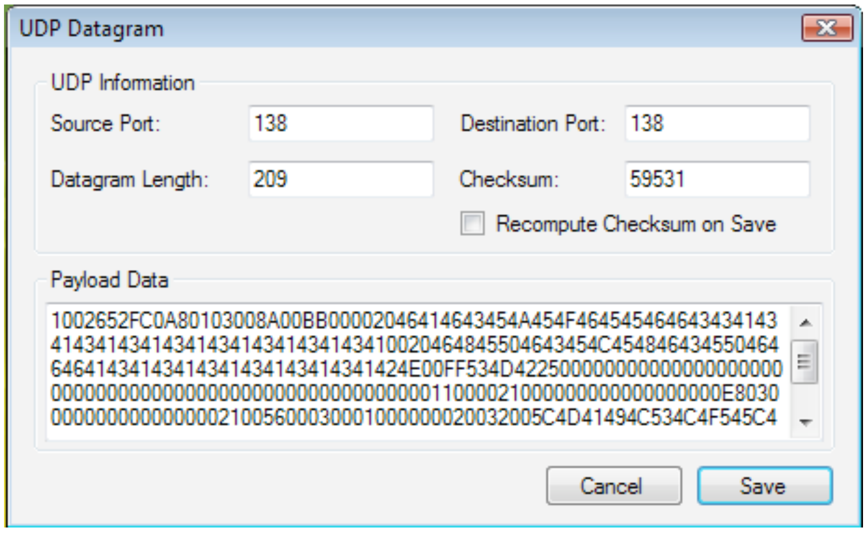
Source Address: 192.168.1.3

Destination Address: 192.168.1.255

Payload

```
008A008A00D1E88B1002652FC0A80103008A00BB000020464146434 54A454F4645454
646434341434143414341434143414341434143410020464845504643454C4548464345
504646464143414341434143414341434142 4E00FF534D4225000000000000000000000000
0000000000000000000000000000000001100002100000000000000000000E8030000000000
000000002100560003000100000000200320 05C4D41494C534C4F545C42524F57534500
010080A903005052494E544552000000000000000000000001000303000000000AA5500
```

Figure 6 - UDP Layer Display



Source Port: 138
Destination Port: 138
Datagram Length: 209
Checksum: 59531
[ ] Recompute Checksum on Save

Payload Data

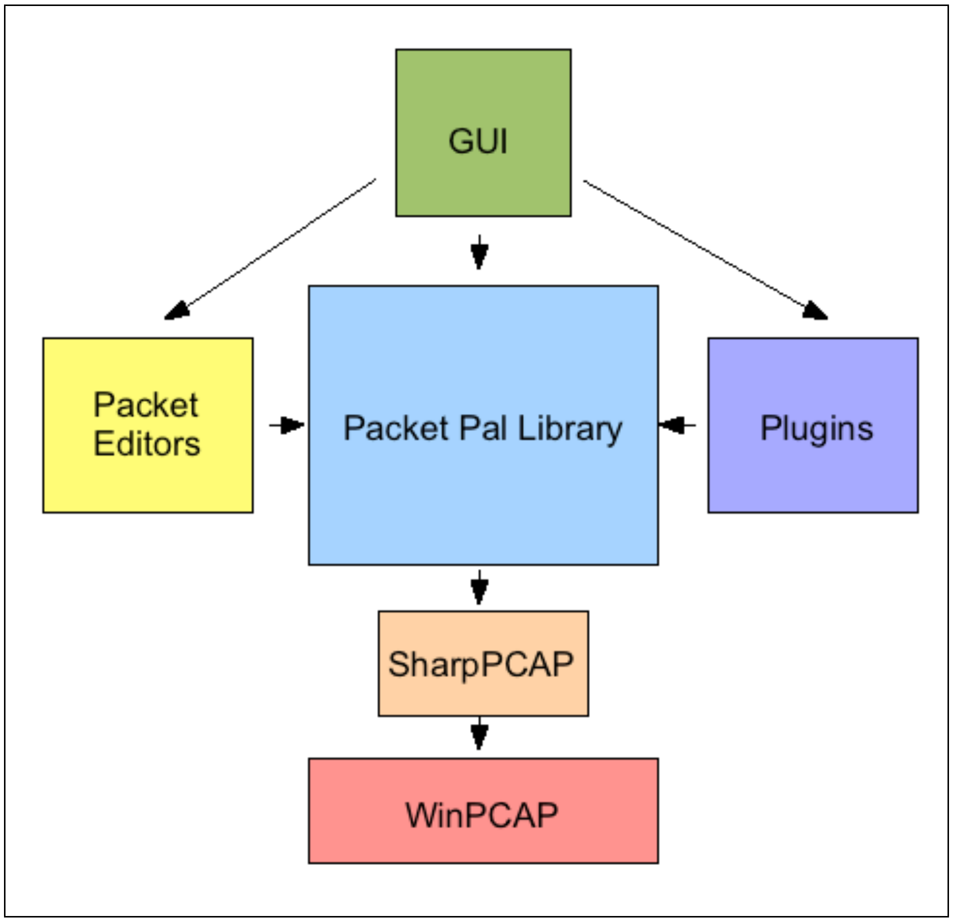
```
1002652FC0A80103008A00BB00002046414643454A454F464545464643434143
4143414341434143414341434143410020464845504643454C454846434550464
6464143414341434143414341434141424E00FF534D42250000000000000000000
00000000000000000000000000000000001100002100000000000000000000E8030
0000000000000002100560003000100000000020032005C4D41494C534C4F545C4
```

While this solution seems to have solved the previous problems, it also generates some difficulties of its own. For instance, the presentation of information in an academic manner is less efficient for constructing numerous packets for something such as a TCP stream. This can make generating larger sets of packets very time-consuming for an application such as penetration testing, which is why the extensibility of Packet Pal is vital.

Program Extension

Packet Pal's modular object-oriented design supports the dynamic loading of several parts of the program including the **GUI**, packet editors, and plug-ins. These features mean the possibilities for Packet Pal are potentially endless. With these extensibility features come some complex, but powerful **APIs** that can provide a solid platform for later development and research. The only constant part of Packet Pal is the core library, which handles all major features and provides a communications channel between all of the other modules. *Figure 7* shows an overview of Packet Pal's modular organization.

Figure 7 - Layout of Packet Pal

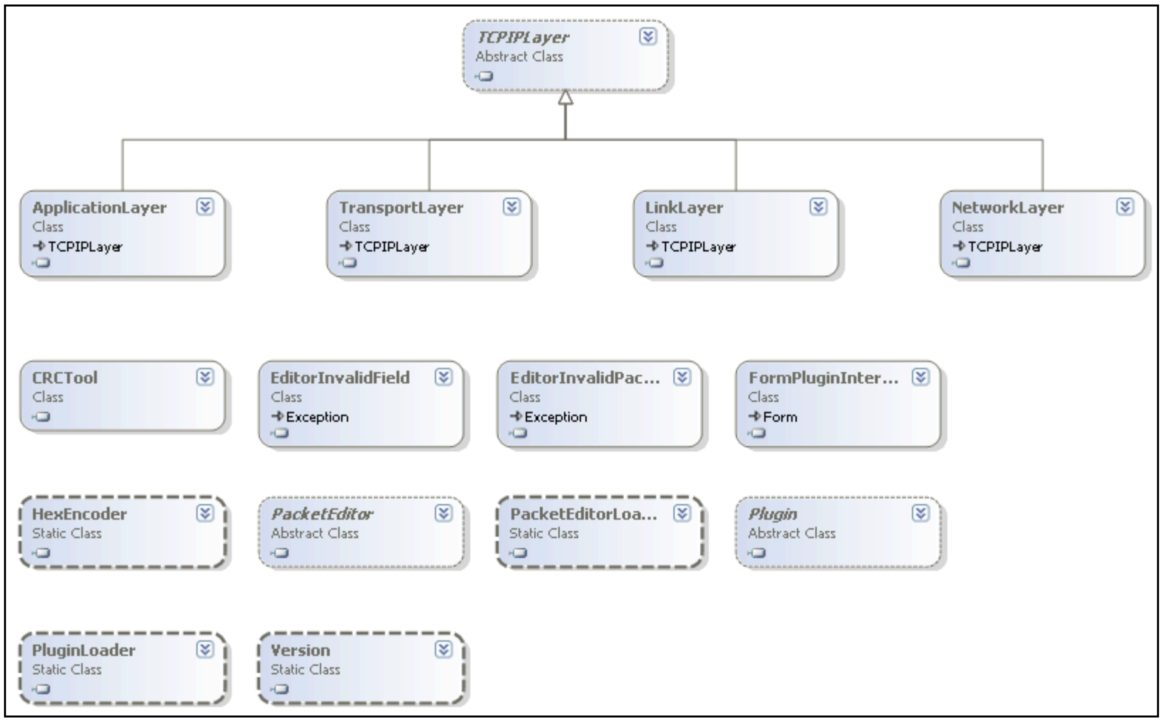


With this type of organization, separate parts of the software can be updated without affecting other areas. For instance: SharpPcap can be upgraded to the latest version, but the various modules of Packet Pal will continue to communicate in the same manner. The concept of independent modules is at the core of the object-oriented programming paradigm. The only caveat to this is that certain third party plug-ins or packet editor modules may require certain versions of other important components to function properly, such as WinPcap and SharpPcap. Those who wish to extend the functionality of Packet Pal can use the available **APIs** in the Packet Pal Library to develop plug-ins, create new **GUIs**, and support the editing of current unsupported network protocols.

Packet Pal Library

The Packet Pal Library is the core of this project. This library provides the interfaces that define the communication between individual pieces of Packet Pal. In addition, tools are provided to complete some network related operations more easily. A simplified class diagram of the Packet Pal Library is shown in *Figure 8*.

Figure 8 - Class Diagram for the Packet Pal Library



Plug-ins

Packet Pal supports the addition of third party plug-ins. A plug-in is a library of code, which extends the functionality of Packet Pal. With the plug-in interface built in to Packet Pal, developers can interact with the core system, including the capture, modification, and broadcast of packets. Plug-ins are loaded automatically at startup, and can be reloaded at any time.

Packet Editors

Packet Editors are the libraries used by Packet Pal to display and modify network packets. Third party developers can create additional editors to allow Packet Pal to natively edit network protocols that are not currently supported. Packet Editors are automatically loaded at startup. Packet Pal must be restarted to reflect newly installed Packet Editors.

Interface

The **GUI** of Packet Pal can be modified or completely replaced. This allows developers to use Packet Pal in situations where it was not originally intended, or where the original interface does not meet the productivity needs of the end users. The current implementation of Packet Pal only includes one interface. The interface is currently used to load and initialize the Packet Pal library.

The Loopback Interface in Windows

The Windows network architecture does not include a standard loopback interface that is usable by WinPcap [20] [2]. The loopback functionality in Windows is implemented in the network stack, rather than by creating an artificial network interface and binding it to the loopback address, which is how UNIX variants accomplish the loopback functionality [19]. Because WinPcap inserts itself between the network device driver layer and the network stack, WinPcap cannot see the standard Windows loopback and will not receive any messages.

Microsoft provides a solution to this problem in the form of the Microsoft LoopBack Interface [11]. By installing this driver and creating a pseudo-device, we can achieve a loopback interface that WinPcap is able to communicate with and monitor. Instructions for installing Microsoft's LoopBack driver can be found in

Appendix A: Packet Pal Help.

This solution is not a complete replacement of the original Windows loopback interface, however, and does not operate in the same manner. For instance, traffic generated by the “ping” command sent to normal loopback addresses such as “localhost” and “127.0.0.1” will never reach WinPcap and Packet Pal because the network stack will process the traffic and it will never reach the driver layer. When the LoopBack driver is installed, a non-routable IP address is automatically assigned to the new interface. Traffic must be sent to this address *from this address* in order to reach WinPcap and Packet Pal. The device resides completely separately from other network devices. With this method, the important requirement of traffic not being broadcast on a real interface is achieved.

Competitive Products

While there are many network packet capturing and analysis tools available, in both the open-source and commercial environments, very few tools exhibit the ability to modify and rebroadcast network packets, a key feature of Packet Pal. This product analysis will look specifically at products, and combinations of products, found to exhibit these important qualities.

Tcprewrite and tcpreplay

Tcprewrite is an open source command line tool based on the PCAP library. Tcprewrite was written by Aaron Turner and released under the BSD license. This tool reads a pcap formatted file of captured packets, performs some specified rewriting operations and writes the resulting packets to a pcap formatted output file.

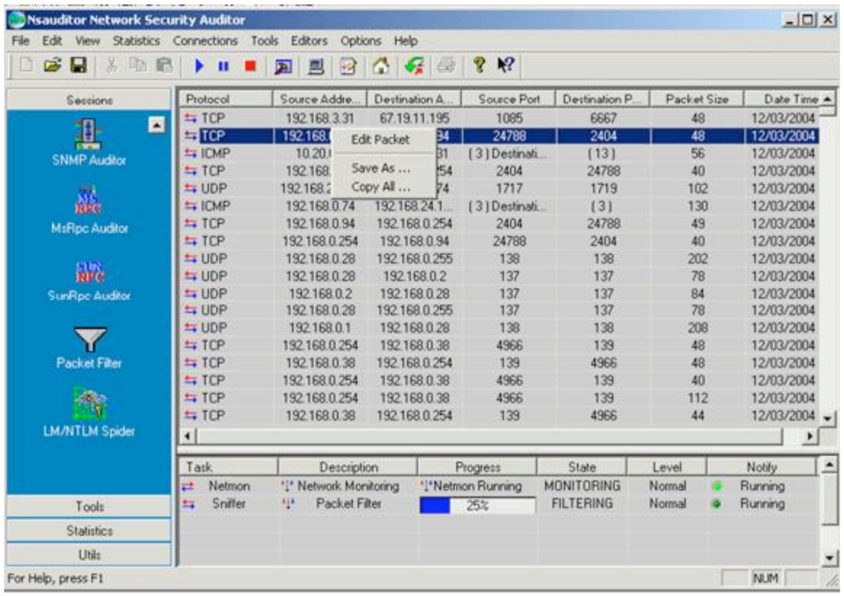
Tcpreplay is another open source command line utility written by Aaron Turner. Tcpreplay reads in a pcap formatted file and broadcasts the packets across a selected network.

While these tools are able to complete the operations of editing packets and re-broadcasting them, there are some obvious limitations. First, there is no included capturing utility. It is assumed that a libpcap-based tool like **tcpdump** or Wireshark will be used to capture the initial traffic and save it to a file. Second, all operations performed by tcprewrite are performed on the entire input file, meaning the detailed editing of an individual packet is impossible with this method. Third, the lack of a **GUI** makes demonstrations with these programs difficult in a classroom environment.

NSAuditor Packet Editor by NsaSoft

NSAuditor is a suite of tools used for network auditing. While there is a packet modification tool, called “Packet Editor” included in the suite, this is not the major focus of the NSAuditor software. With the enormous number of auditing tools included in NSAuditor, a user is able to capture, edit, and broadcast packets. NSAuditor also includes a tool called Traffic Emulator, which easily generates certain types of network traffic. NSAuditor’s packet capture interface is shown in *Figure 9* and its Packet Editor is shown in *Figure 10*. All screen shots of NSAuditor are property of NsaSoft, LLC.

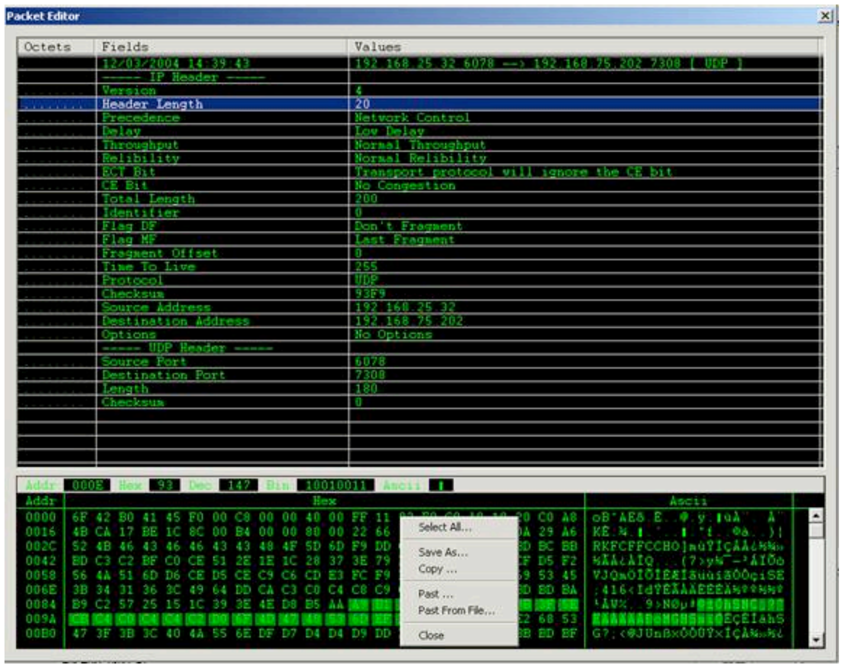
Figure 9 - NSAuditor's Packet Capture Interface



Protocol	Source Addre...	Destination A...	Source Port	Destination P...	Packet Size	Date Time
TCP	192.168.3.31	67.19.11.195	1085	6667	48	12/03/2004
TCP	192.168.	34	24788	2404	48	12/03/2004
ICMP	10.20.	31	(3) Destinati...	(13)	56	12/03/2004
TCP	192.168.	54	2404	24788	40	12/03/2004
UDP	192.168.2	74	1717	1719	102	12/03/2004
ICMP	192.168.0.74	192.168.24.1...	(3) Destinati...	(3)	130	12/03/2004
TCP	192.168.0.94	192.168.0.254	2404	24788	49	12/03/2004
TCP	192.168.0.254	192.168.0.94	24788	2404	40	12/03/2004
UDP	192.168.0.28	192.168.0.255	138	138	202	12/03/2004
UDP	192.168.0.28	192.168.0.2	137	137	78	12/03/2004
UDP	192.168.0.2	192.168.0.28	137	137	84	12/03/2004
UDP	192.168.0.28	192.168.0.255	137	137	78	12/03/2004
UDP	192.168.0.1	192.168.0.28	138	138	208	12/03/2004
TCP	192.168.0.254	192.168.0.38	4966	139	48	12/03/2004
TCP	192.168.0.38	192.168.0.254	139	4966	48	12/03/2004
TCP	192.168.0.254	192.168.0.38	4966	139	40	12/03/2004
TCP	192.168.0.254	192.168.0.38	4966	139	112	12/03/2004
TCP	192.168.0.38	192.168.0.254	139	4966	44	12/03/2004

Task	Description	Progress	State	Level	Notify
Netmon	Network Monitoring	Netmon Running	MONITORING	Normal	● Running
Sniffer	Packet Filter	25%	FILTERING	Normal	● Running

Figure 10 - NSAuditor's Packet Editor Interface



Octets	Fields	Values
	12/03/2004 14:39:43	192.168.25.32 6078 ---> 192.168.75.202 7308 [UDP]
	----- IP Header -----	
....	Version	4
....	Header Length	20
....	Precedence	Network Control
....	Delay	Low Delay
....	Throughput	Normal Throughput
....	Relibility	Normal Relibility
....	ECT Bit	Transport protocol will ignore the CE bit
....	CE Bit	No Congestion
....	Total Length	200
....	Identifier	0
....	Flag DF	Don't Fragment
....	Flag MF	Last Fragment
....	Fragment Offset	0
....	Time To Live	255
....	Protocol	UDP
....	Checksum	93F9
....	Source Address	192.168.25.32
....	Destination Address	192.168.75.202
....	Options	No Options
	----- UDP Header -----	
....	Source Port	6078
....	Destination Port	7308
....	Length	180
....	Checksum	0

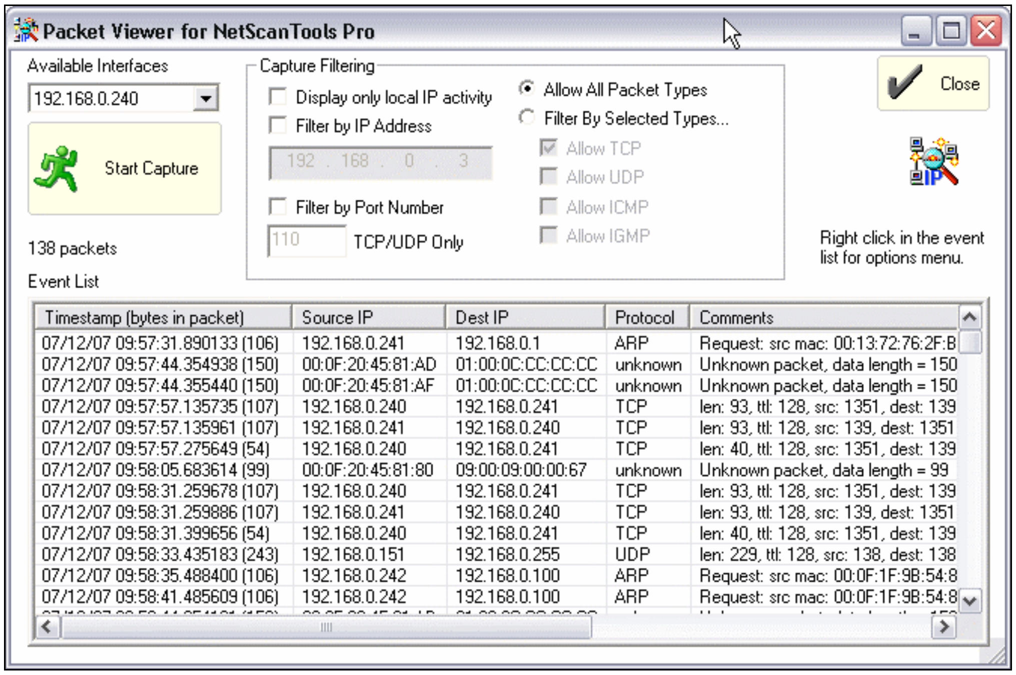
```
Addr 000E  Hex 93  Dec 147  Bin 10010011  Ascii
Addr                  Hex
0000  6F 42 B0 41 45 F0 00 C8 00 00 40 00 FF 11 [illegible] 20 C0 A8
0016  4B CA 17 BE 1C 8C 00 B4 00 00 80 00 22 66 [illegible] 0A 29 A6
002C  52 4B 46 43 46 46 43 43 48 4F 5D 6D F9 DD [illegible] BD BC BB
0042  BD C3 C2 BF C0 CE 51 2E 1E 1C 28 37 3E 79 [illegible] CF D5 F2
0058  56 4A 51 6D D6 CE D5 CE C9 C6 CD E3 FC F9 [illegible] 59 53 45
006E  3B 34 31 36 3C 49 64 DD CA C3 C0 C4 C8 C9 [illegible] BD BD BA
0084  B9 C2 57 25 15 1C 39 3E 4E D8 B5 AA [illegible]
009A  [illegible] E2 68 53
00B0  47 3F 3B 3C 40 4A 55 6E DF D7 D4 D4 D9 DD [illegible] BB BD BF
```

The NSAuditor suite is a very useful commercial penetration-testing tool, and is reasonably priced. The most obvious shortcoming of NSAuditor lies in its lack of a public plug-in **API**, which would allow third parties to develop extensions. Although many commercial products support third party extensions, Adobe Photoshop for example, there is no indication that NsaSoft will pursue this option in the near future.

NetScan Tools Pro by NorthWest Performance Software

NetScan Tools Pro is a commercial suite of tools designed for penetration testing. Tools are provided for capturing traffic, generating TCP and UDP traffic, and launching preconfigured exploits. The capture window and detailed view, shown in *Figure 11* and *Figure 12* respectively, are similar in design to Wireshark. The Packet Generator, included with NetScan Tools Pro is shown in *Figure 13*. All screen shots of NetScan Tools Pro are property of Northwest Performance Software, Inc.

Figure 11 - NetScan Tools Pro Capture Window



192.168.0.240

Display only local IP activity [ ]
Filter by IP Address [ ]
192 . 168 . 0 . 3
Filter by Port Number [ ]
110 TCP/UDP Only

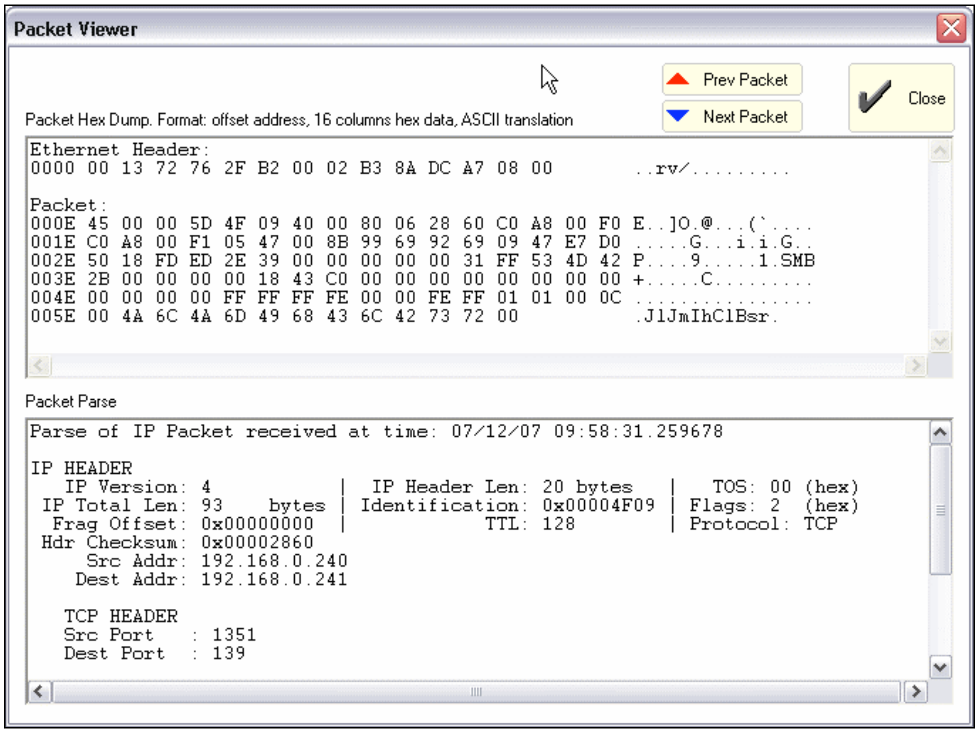
[✓ Allow All Packet Types]
Filter By Selected Types...
Allow TCP [✓]
Allow UDP [ ]
Allow ICMP [ ]
Allow IGMP [ ]

Right click in the event list for options menu.

138 packets

Timestamp (bytes in packet)	Source IP	Dest IP	Protocol	Comments
07/12/07 09:57:31.890133 (106)	192.168.0.241	192.168.0.1	ARP	Request: src mac: 00:13:72:76:2F:B
07/12/07 09:57:44.354938 (150)	00:0F:20:45:81:AD	01:00:0C:CC:CC:CC	unknown	Unknown packet, data length = 150
07/12/07 09:57:44.355440 (150)	00:0F:20:45:81:AF	01:00:0C:CC:CC:CC	unknown	Unknown packet, data length = 150
07/12/07 09:57:57.135735 (107)	192.168.0.240	192.168.0.241	TCP	len: 93, ttl: 128, src: 1351, dest: 139
07/12/07 09:57:57.135961 (107)	192.168.0.241	192.168.0.240	TCP	len: 93, ttl: 128, src: 139, dest: 1351
07/12/07 09:57:57.275649 (54)	192.168.0.240	192.168.0.241	TCP	len: 40, ttl: 128, src: 1351, dest: 139
07/12/07 09:58:05.683614 (99)	00:0F:20:45:81:80	09:00:09:00:00:67	unknown	Unknown packet, data length = 99
07/12/07 09:58:31.259678 (107)	192.168.0.240	192.168.0.241	TCP	len: 93, ttl: 128, src: 1351, dest: 139
07/12/07 09:58:31.259886 (107)	192.168.0.241	192.168.0.240	TCP	len: 93, ttl: 128, src: 139, dest: 1351
07/12/07 09:58:31.399656 (54)	192.168.0.240	192.168.0.241	TCP	len: 40, ttl: 128, src: 1351, dest: 139
07/12/07 09:58:33.435183 (243)	192.168.0.151	192.168.0.255	UDP	len: 229, ttl: 128, src: 138, dest: 138
07/12/07 09:58:35.488400 (106)	192.168.0.242	192.168.0.100	ARP	Request: src mac: 00:0F:1F:9B:54:8
07/12/07 09:58:41.485609 (106)	192.168.0.242	192.168.0.100	ARP	Request: src mac: 00:0F:1F:9B:54:8

Figure 12 - NetScan Tools Pro Packet Detail



Packet Hex Dump. Format: offset address, 16 columns hex data, ASCII translation

```
Ethernet Header:
0000 00 13 72 76 2F B2 00 02 B3 8A DC A7 08 00        ..rv/.........

Packet:
000E 45 00 00 5D 4F 09 40 00 80 06 28 60 C0 A8 00 F0 E..]O.@...(`....
001E C0 A8 00 F1 05 47 00 8B 99 69 92 69 09 47 E7 D0 .....G...i.i.G..
002E 50 18 FD ED 2E 39 00 00 00 00 00 31 FF 53 4D 42 P....9.....1.SMB
003E 2B 00 00 00 00 18 43 C0 00 00 00 00 00 00 00 00 +.....C.........
004E 00 00 00 00 FF FF FF FE 00 00 FE FF 01 01 00 0C ................
005E 00 4A 6C 4A 6D 49 68 43 6C 42 73 72 00           .JlJmIhClBsr.
```

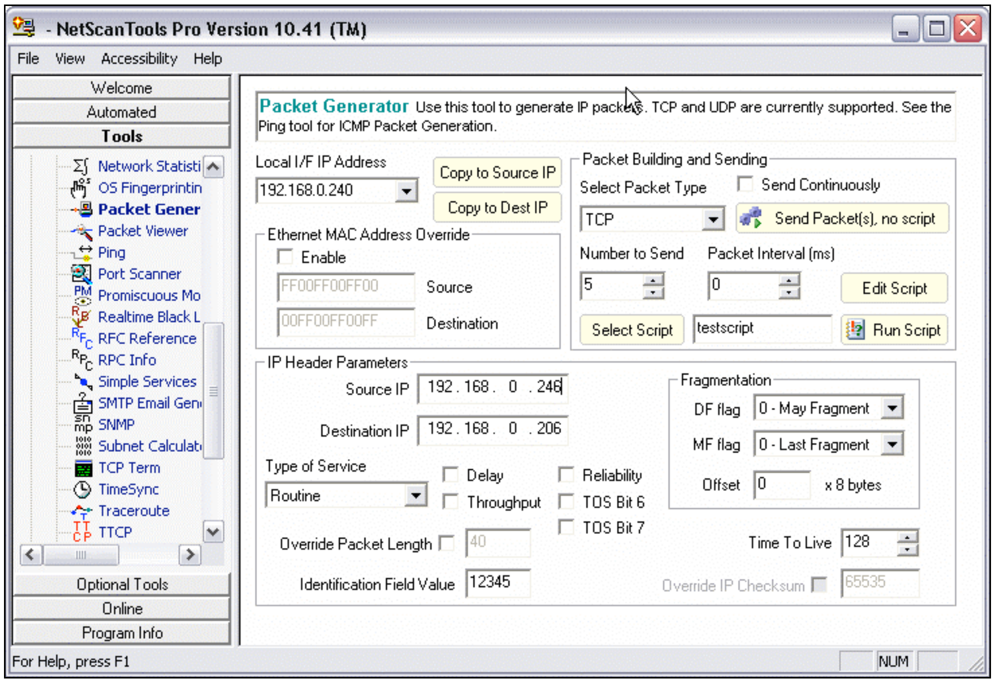
Packet Parse

```
Parse of IP Packet received at time: 07/12/07 09:58:31.259678

IP HEADER
   IP Version: 4          |  IP Header Len: 20 bytes    |   TOS: 00 (hex)
 IP Total Len: 93    bytes | Identification: 0x00004F09 | Flags: 2  (hex)
  Frag Offset: 0x00000000  |            TTL: 128        | Protocol: TCP
 Hdr Checksum: 0x00002860
     Src Addr: 192.168.0.240
    Dest Addr: 192.168.0.241

   TCP HEADER
   Src Port   : 1351
   Dest Port  : 139
```

Figure 13 - NetScan Tools Pro Packet Generator



Packet Generator Use this tool to generate IP packets. TCP and UDP are currently supported. See the Ping tool for ICMP Packet Generation.

Local I/F IP Address: 192.168.0.240

Ethernet MAC Address Override
- ☐ Enable
- Source: FF00FF00FF00
- Destination: 00FF00FF00FF

Packet Building and Sending
- Select Packet Type: TCP
- ☐ Send Continuously
- Number to Send: 5
- Packet Interval (ms): 0
- testscript

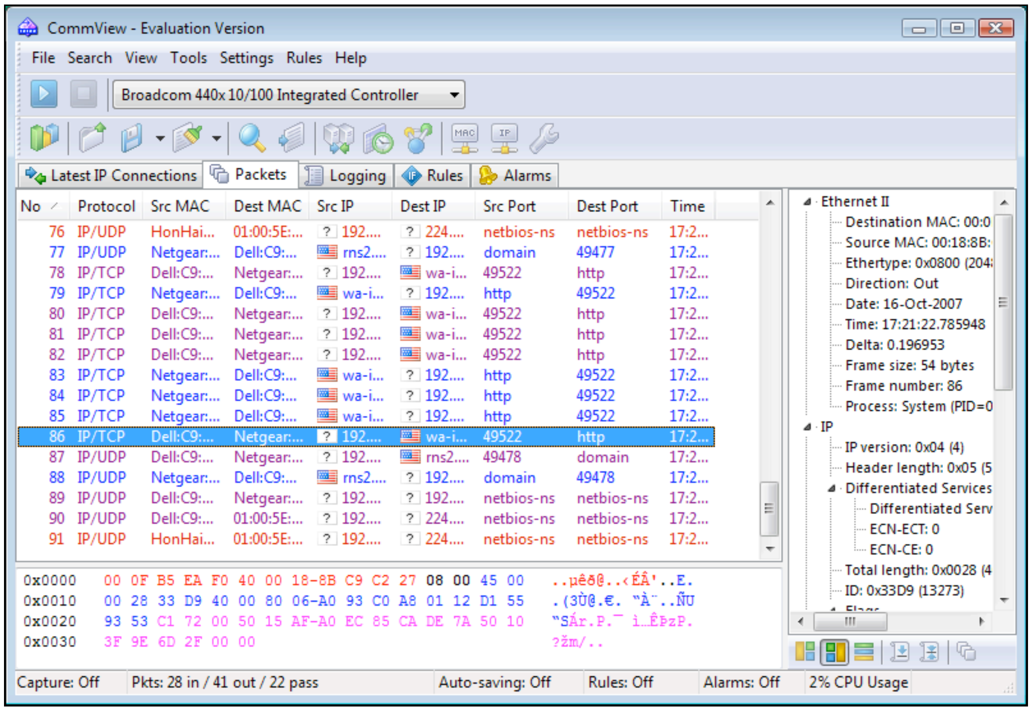
IP Header Parameters
- Source IP: 192 . 168 . 0 . 246
- Destination IP: 192 . 168 . 0 . 206
- Type of Service: Routine
- ☐ Delay ☐ Reliability
- ☐ Throughput ☐ TOS Bit 6
- ☐ TOS Bit 7
- Override Packet Length ☐ 40
- Identification Field Value: 12345
- Fragmentation
  - DF flag: 0 - May Fragment
  - MF flag: 0 - Last Fragment
  - Offset: 0 x 8 bytes
- Time To Live: 128
- Override IP Checksum ☐ 65535

The Packet Generator utility is very useful for generating complete streams of TCP communication, specifically for penetration testing. The important feature missing from NestScan Tools Pro is the ability to edit captured traffic. While NetScan Tools Pro allows the user to generate and broadcast traffic with the Packet Generator, there is no provided tool for the raw manipulation of previously captured traffic. Another missing feature, as with other commercial tools like NSAuditor, is the ability to create third party extensions through a public **API**.

CommView by TamoSoft

CommView is a commercial network penetration-testing tool. Like NSAuditor and NetScan Tools Pro, CommView provides tools for capturing traffic. The capture interface, like NetScan Tools Pro, resembles the classic Wireshark interface. Commview also provides a Packet Generator, used to create one or more packets for broadcast. Unlike NetScan Tools Pro's generator, the creation process in CommView is more in-depth and does not easily allow for the creation of streams consisting of multiple packets. CommView's capture interface and detailed packet view interface are shown in *Figure 14* and *Figure 15* respectively. All screen shots of CommView are property of TamoSoft.

Figure 14 - CommView Packet Information


CommView - Evaluation Version
File Search View Tools Settings Rules Help
Broadcom 440x 10/100 Integrated Controller
MAC
IP
Latest IP Connections
Packets
Logging
Rules
Alarms

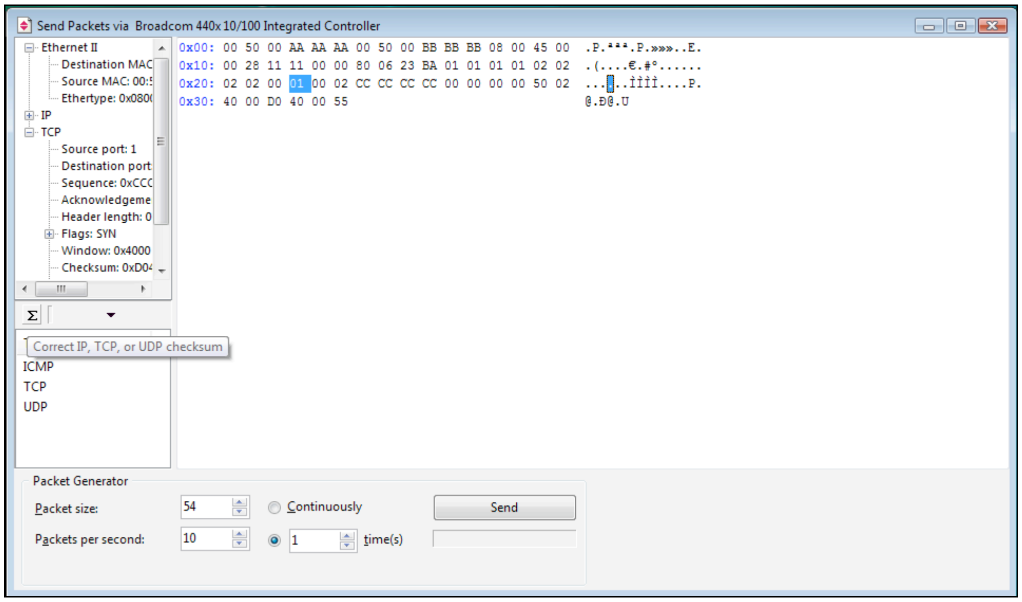
No	Protocol	Src MAC	Dest MAC	Src IP	Dest IP	Src Port	Dest Port	Time
76	IP/UDP	HonHai...	01:00:5E:...	? 192....	? 224....	netbios-ns	netbios-ns	17:2...
77	IP/UDP	Netgear:...	Dell:C9:...	rns2....	? 192....	domain	49477	17:2...
78	IP/TCP	Dell:C9:...	Netgear:...	? 192....	wa-i...	49522	http	17:2...
79	IP/TCP	Netgear:...	Dell:C9:...	wa-i...	? 192....	http	49522	17:2...
80	IP/TCP	Dell:C9:...	Netgear:...	? 192....	wa-i...	49522	http	17:2...
81	IP/TCP	Dell:C9:...	Netgear:...	? 192....	wa-i...	49522	http	17:2...
82	IP/TCP	Dell:C9:...	Netgear:...	? 192....	wa-i...	49522	http	17:2...
83	IP/TCP	Netgear:...	Dell:C9:...	wa-i...	? 192....	http	49522	17:2...
84	IP/TCP	Netgear:...	Dell:C9:...	wa-i...	? 192....	http	49522	17:2...
85	IP/TCP	Netgear:...	Dell:C9:...	wa-i...	? 192....	http	49522	17:2...
86	IP/TCP	Dell:C9:...	Netgear:...	? 192....	wa-i...	49522	http	17:2...
87	IP/UDP	Dell:C9:...	Netgear:...	? 192....	rns2....	49478	domain	17:2...
88	IP/UDP	Netgear:...	Dell:C9:...	rns2....	? 192....	domain	49478	17:2...
89	IP/UDP	Dell:C9:...	Netgear:...	? 192....	? 192....	netbios-ns	netbios-ns	17:2...
90	IP/UDP	Dell:C9:...	01:00:5E:...	? 192....	? 224....	netbios-ns	netbios-ns	17:2...
91	IP/UDP	HonHai...	01:00:5E:...	? 192....	? 224....	netbios-ns	netbios-ns	17:2...

```
0x0000   00 0F B5 EA F0 40 00 18-8B C9 C2 27 08 00 45 00   ..µêð@..‹ÉÂ'..E.
0x0010   00 28 33 D9 40 00 80 06-A0 93 C0 A8 01 12 D1 55   .(3Ù@.€. “À¨..ÑU
0x0020   93 53 C1 72 00 50 15 AF-A0 EC 85 CA DE 7A 50 10   “SÁr.P.¯ ì…ÊÞzP.
0x0030   3F 9E 6D 2F 00 00                                 ?žm/..
```

```
Ethernet II
    Destination MAC: 00:0
    Source MAC: 00:18:8B:
    Ethertype: 0x0800 (204
    Direction: Out
    Date: 16-Oct-2007
    Time: 17:21:22.785948
    Delta: 0.196953
    Frame size: 54 bytes
    Frame number: 86
    Process: System (PID=0
IP
    IP version: 0x04 (4)
    Header length: 0x05 (5
    Differentiated Services
        Differentiated Serv
        ECN-ECT: 0
        ECN-CE: 0
    Total length: 0x0028 (4
    ID: 0x33D9 (13273)
```

Capture: Off | Pkts: 28 in / 41 out / 22 pass | Auto-saving: Off | Rules: Off | Alarms: Off | 2% CPU Usage

Figure 15 - CommView Packet Generator


Send Packets via Broadcom 440x 10/100 Integrated Controller

```
Ethernet II
    Destination MAC
    Source MAC: 00:5
    Ethertype: 0x080(
IP
TCP
    Source port: 1
    Destination port
    Sequence: 0xCCC
    Acknowledgeme
    Header length: 0
    Flags: SYN
    Window: 0x4000
    Checksum: 0xD04
```

```
0x00:  00 50 00 AA AA AA 00 50 00 BB BB BB 08 00 45 00  .P.ªªª.P.»»»..E.
0x10:  00 28 11 11 00 00 80 06 23 BA 01 01 01 01 02 02  .(....€.#º......
0x20:  02 02 00 01 00 02 CC CC CC CC 00 00 00 00 50 02  ......ÌÌÌÌ....P.
0x30:  40 00 D0 40 00 55                                @.Ð@.U
```

Correct IP, TCP, or UDP checksum

ICMP
TCP
UDP

Packet Generator

Packet size: 54
Packets per second: 10
Continuously
[✓ 1] time(s)

Where CommView separates itself from the other commercial software is with its extensible plug-in **API**. Third party plug-ins can be written to access captured packets. Although CommView lacks the ability to edit captured packets, this feature could theoretically be added as a third party extension. Examples of how to write plug-ins for CommView are given on the TamoSoft web site.

Product Comparison

Only two of these product suites, tcprewrite/tcpreplay and NSAuditor, provide a complete solution to the tasks of capturing, editing, and re-broadcasting network communications. NetScan Tools Pro and CommView both lack the ability to edit captured network packets, although the possibility exists for a third party plug-in to be written for CommView to complete this task. *Table 1* highlights some of more important functionality in these programs with an emphasis on Packet Pal's key features.

Table 1 - Product Difference Matrix

	Packet Pal	tcprewrite	NSAuditor	NetScan Tools Pro	CommView
Graphical Capturing	•		•	•	•
Graphical Editing	•		•		
Graphical Broadcast	•		•	•	•
Extensible Packet Editing	•	•			
Plug-in Interface	•	•			•

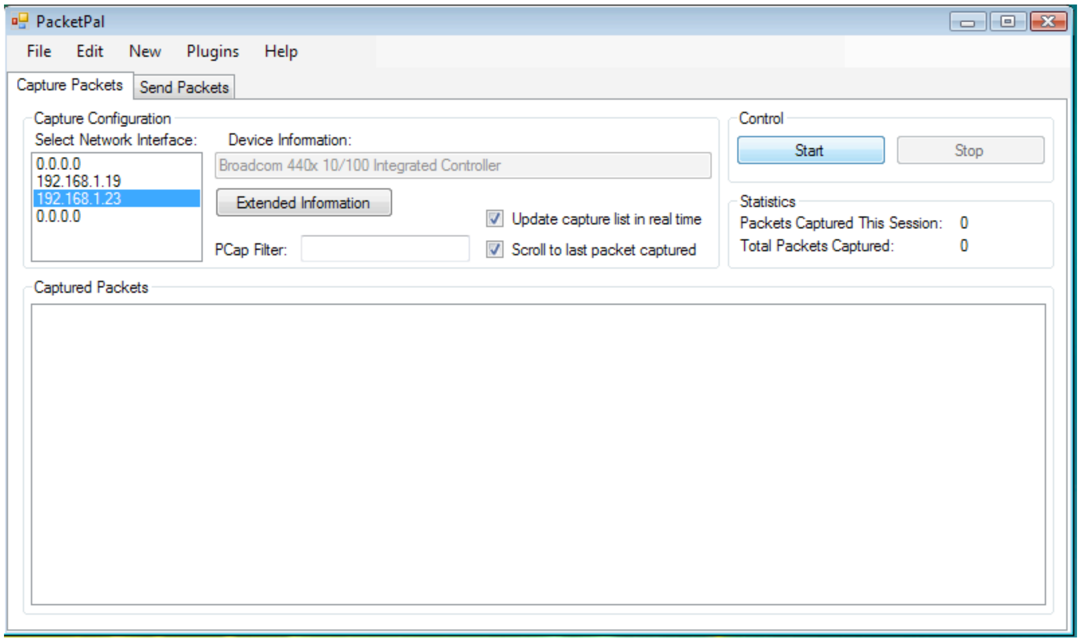
Requires libpcap	•	•			
Runs in Microsoft Windows	•		•	•	•
Runs in Linux		•			
Open-Source	•	•			

Demonstration of Capture, Edit, and Broadcast

The main screen for Packet Pal is displayed in *Figure 16*. This product demonstration will show the capture, editing, and re-broadcasting of a packet.

Assuming the host computer meets the system requirements, the first step is to capture a packet from the host machine's network. To do this, select the "Capture Packets" tab, select the interface to capture from within the given list, and press the "Start" button. If the host network is quiet, using the "ping" command from the host machine's console or loading a web page in a web browser on the host machine can generate traffic.

Figure 16 - Main Packet Pal Window



Select Network Interface:
0.0.0.0
192.168.1.19
192.168.1.23
0.0.0.0

Device Information: Broadcom 440x 10/100 Integrated Controller

PCap Filter:

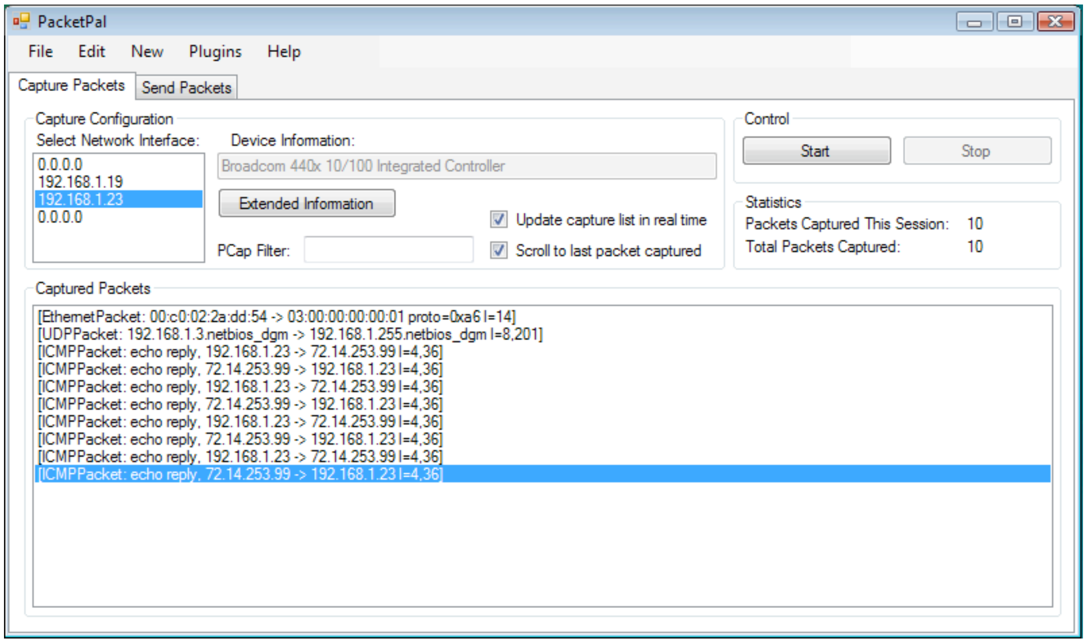
☑ Update capture list in real time

☑ Scroll to last packet captured

Statistics
Packets Captured This Session: 0
Total Packets Captured: 0

Once the desired packets are captured, press the "Stop" button to view the captured packets. *Figure 17* shows a list of captured packets in Packet Pal.

Figure 17 - Captured Packets



Select Network Interface: 0.0.0.0 / 192.168.1.19 / 192.168.1.23 / 0.0.0.0

Device Information: Broadcom 440x 10/100 Integrated Controller

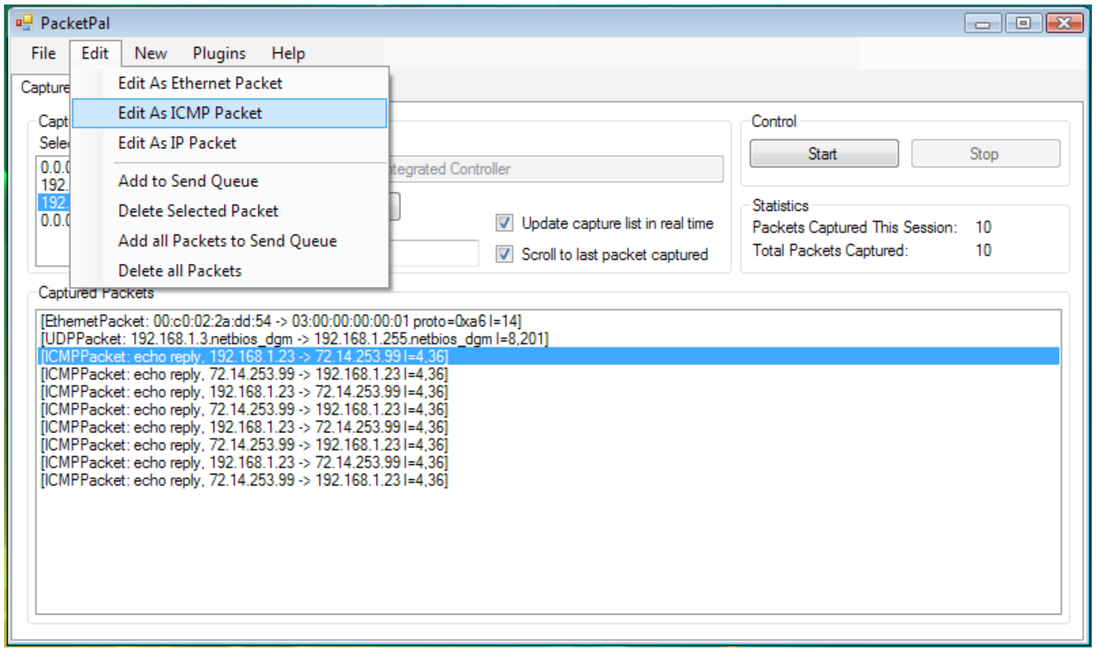
☑ Update capture list in real time
☑ Scroll to last packet captured

Packets Captured This Session: 10
Total Packets Captured: 10

```
[EthernetPacket: 00:c0:02:2a:dd:54 -> 03:00:00:00:00:01 proto=0xa6 l=14]
[UDPPacket: 192.168.1.3.netbios_dgm -> 192.168.1.255.netbios_dgm l=8,201]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
```

To edit a packet, select it by left-clicking on it once. From the "Edit" menu, or by right-clicking on the packet, select the protocol layer of this packet that is desired. *Figure 18* shows the "Edit" menu as it pertains to the currently selected ICMP Packet.

Figure 18 - Editing a Packet



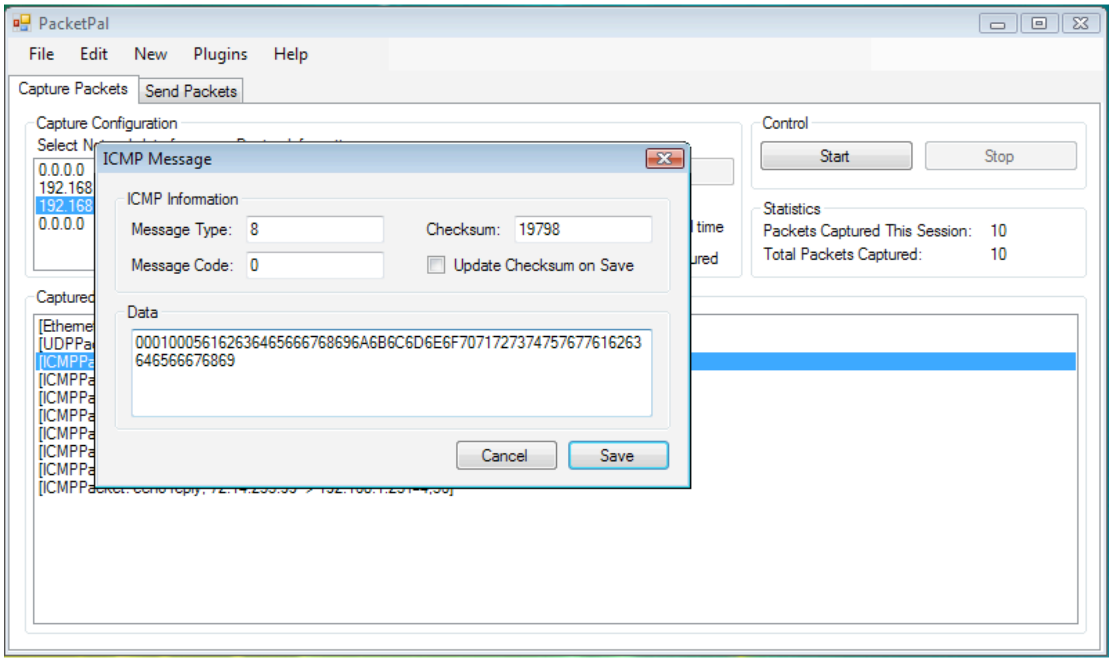
☑ Update capture list in real time
☑ Scroll to last packet captured

Packets Captured This Session: 10
Total Packets Captured: 10

```
[EthernetPacket: 00:c0:02:2a:dd:54 -> 03:00:00:00:00:01 proto=0xa6 l=14]
[UDPPacket: 192.168.1.3.netbios_dgm -> 192.168.1.255.netbios_dgm l=8,201]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
[ICMPPacket: echo reply, 72.14.253.99 -> 192.168.1.23 l=4,36]
```

The interface for editing the selected protocol will appear with the packet's information. Edit the information as desired and click the “Save” button to finalize the changes. The main window is then shown, with the packet’s information updated. *Figure 19* shows the ICMP Packet Editor editing the ICMP Packet selected in *Figure 18*.

Figure 19 - Editing the ICMP Layer



ICMP Message

ICMP Information

Message Type: 8

Message Code: 0

Checksum: 19798

Update Checksum on Save [ ]

Data

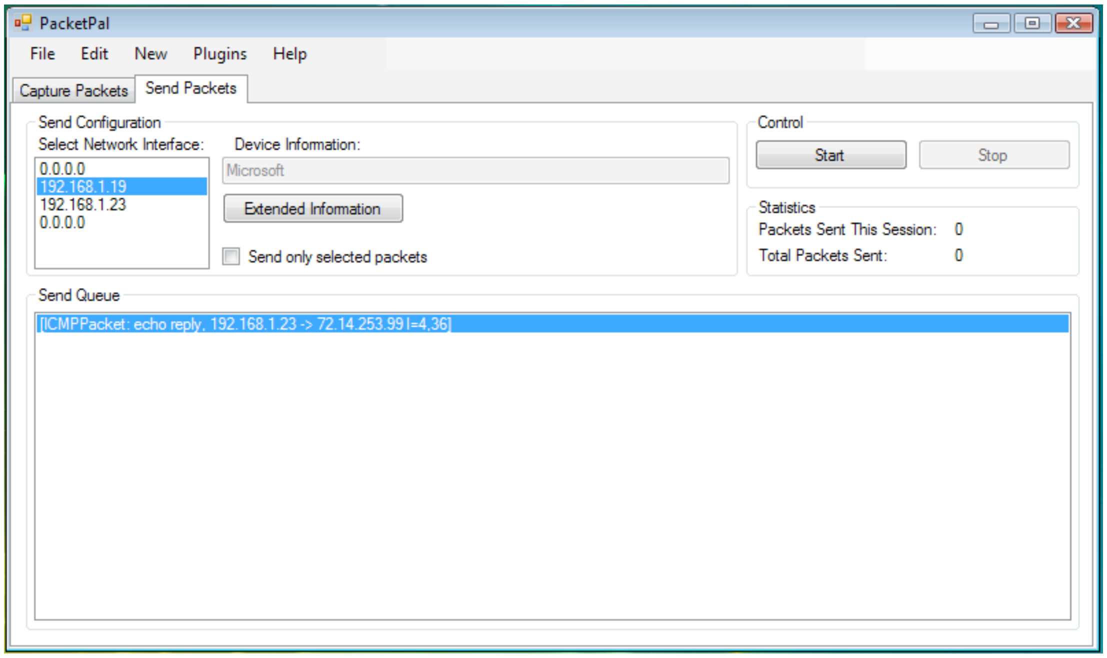
```
000100056162636465666768696A6B6C6D6E6F7071727374757677616263
646566676869
```

Packets Captured This Session: 10

Total Packets Captured: 10

To re-broadcast this packet, it needs to be added to the “Send Queue.” To do this, select the packet by left-clicking on it once. From the “Edit” menu, or by right-clicking on the packet, select the option “Add to Send Queue.” The packet is now in the Send Queue. To view it, click the “Send Packets” tab. *Figure 20* shows the previously selected ICMP packet in the “Send Queue.”

Figure 20 - The Send Queue



0.0.0.0
192.168.1.19
192.168.1.23
0.0.0.0

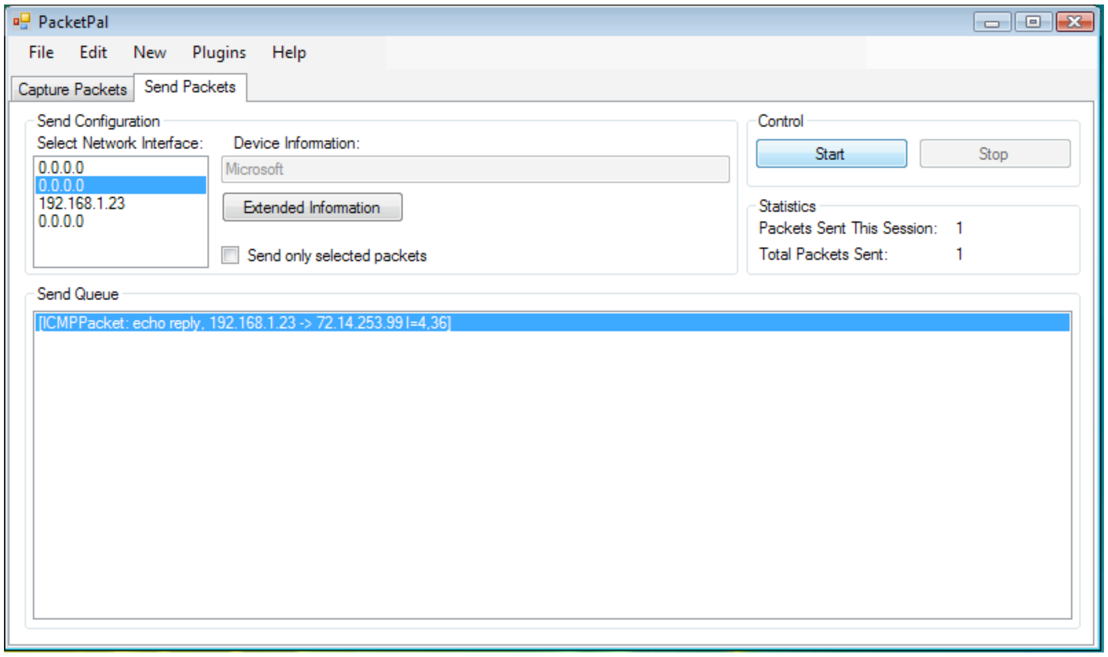
Microsoft

Packets Sent This Session: 0
Total Packets Sent: 0

[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]

To begin the broadcast of all packets in the Send Queue, select the desired network interface to broadcast on from the list, and then press the “Start” button. The “Stop” button can be pressed during the sending process to halt the progression through the Send Queue. *Figure 21* shows Packet Pal after having sent the ICMP packet from the Send Queue. Notice that the “Packets Sent This Session” and “Total Packets Sent” fields have been incremented to reflect the broadcast.

Figure 21 – Completed Sending Packets



0.0.0.0
0.0.0.0
192.168.1.23
0.0.0.0

Device Information: Microsoft

Send only selected packets

Packets Sent This Session: 1
Total Packets Sent: 1

```
[ICMPPacket: echo reply, 192.168.1.23 -> 72.14.253.99 l=4,36]
```

This completes the demonstration of a simple capture, edit, and broadcast scenario. Any other scenarios can be constructed from these simple instructions.

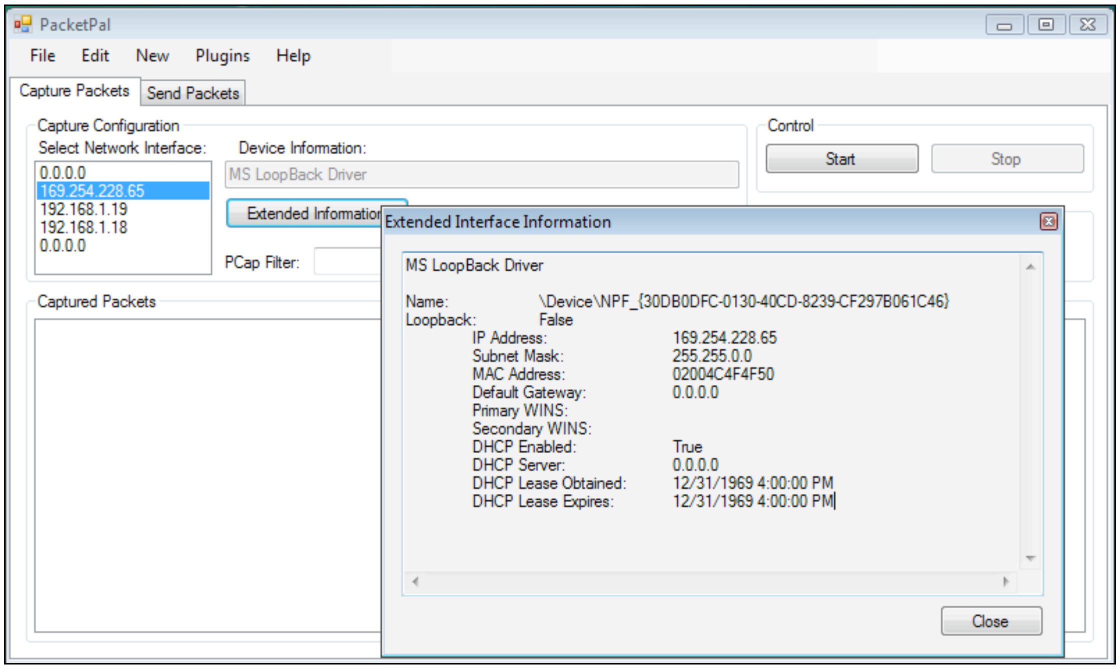
Demonstration Using the Microsoft LoopBack Interface

As mentioned previously, the loopback interface in Windows is not implemented in the same manner as with most UNIX variants. This demonstration assumes the Microsoft LoopBack interface has been installed as described in

Appendix A: Packet Pal Help.

The first step is to open up Packet Pal and verify that the Microsoft LoopBack driver is working properly. There should be an interface with a non-routable IP address. In *Figure 22*, the address for the loopback Interface is “169.254.228.65.”

Figure 22 – Packet Pal with The Microsoft LoopBack Driver



```
0.0.0.0
169.254.228.65
192.168.1.19
192.168.1.18
0.0.0.0
```

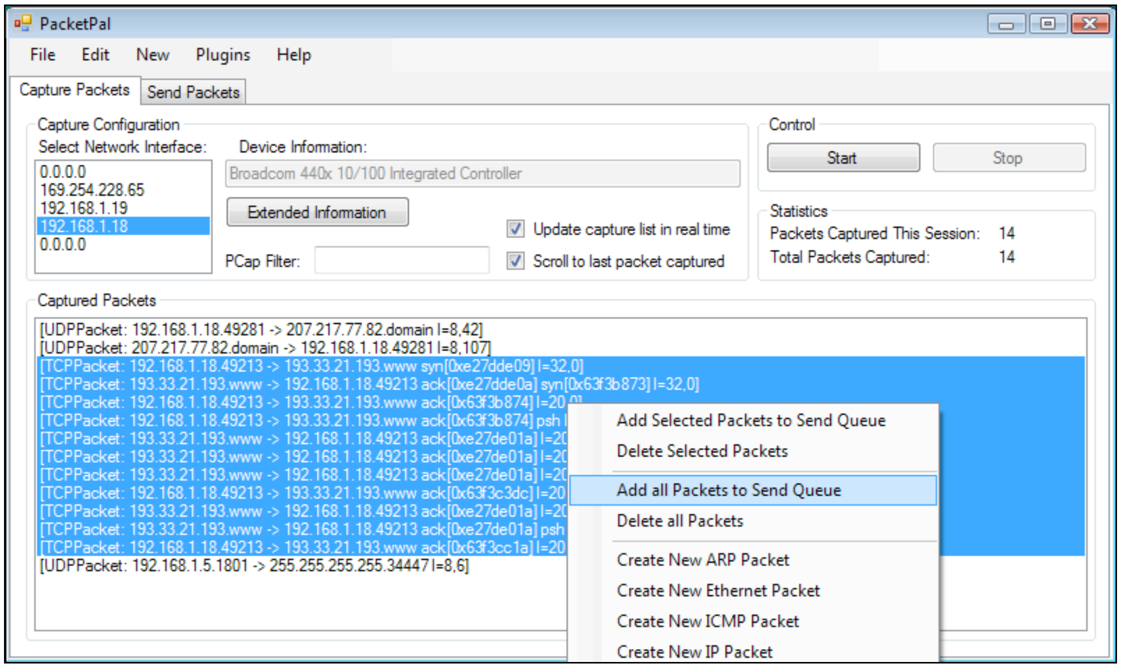
MS LoopBack Driver

```
MS LoopBack Driver

Name:           \Device\NPF_{30DB0DFC-0130-40CD-8239-CF297B061C46}
Loopback:       False
        IP Address:             169.254.228.65
        Subnet Mask:            255.255.0.0
        MAC Address:            02004C4F4F50
        Default Gateway:        0.0.0.0
        Primary WINS:
        Secondary WINS:
        DHCP Enabled:           True
        DHCP Server:            0.0.0.0
        DHCP Lease Obtained:    12/31/1969 4:00:00 PM
        DHCP Lease Expires:     12/31/1969 4:00:00 PM
```

As was stated earlier, a normal “ping” command will not be able to generate traffic on this interface, so traffic needs to be generated elsewhere or from scratch. The simplest way to do this is to select an active interface, “192.168.1.18” in this case, and capture some traffic generated by another program, such as the “ping” command or a web browser. Once the traffic is captured, it needs to be moved to the “Send Queue.” *Figure 23* shows captured packets being moved to the Send Queue.

Figure 23 – Move Packets to Send Queue



Select Network Interface: 0.0.0.0 / 169.254.228.65 / 192.168.1.19 / 192.168.1.18 / 0.0.0.0

Device Information: Broadcom 440x 10/100 Integrated Controller

PCap Filter:

[✓] Update capture list in real time

[✓] Scroll to last packet captured

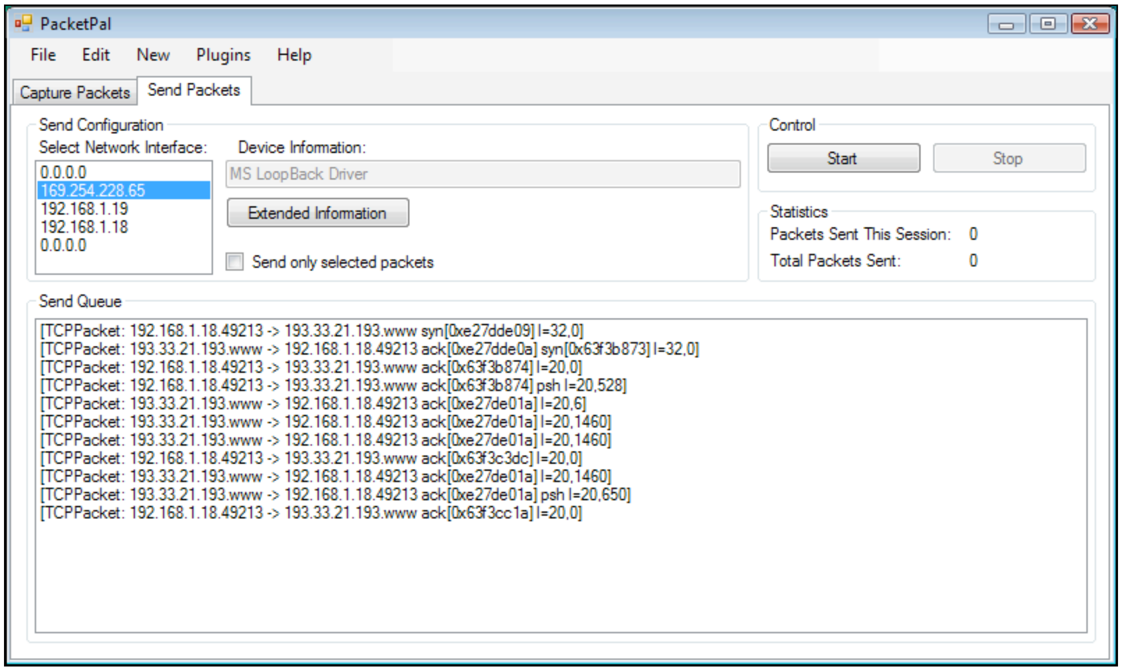
Statistics
Packets Captured This Session: 14
Total Packets Captured: 14

Captured Packets

```
[UDPPacket: 192.168.1.18.49281 -> 207.217.77.82.domain l=8,42]
[UDPPacket: 207.217.77.82.domain -> 192.168.1.18.49281 l=8,107]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www syn[0xe27dde09] l=32,0]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27dde0a] syn[0x63f3b873] l=32,0]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] l=20,0
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] psh
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3c3dc] l=20
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] psh
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3cc1a] l=20
[UDPPacket: 192.168.1.5.1801 -> 255.255.255.255.34447 l=8,6]
```

On the "Send Packets" tab, select the MS LoopBack Driver as the target for broadcasting these packets. It is not necessary to start the sending process yet. In fact, to view the results, it will need to be done later. Pressing the "Send" button at this point, however, will not disrupt the tutorial. *Figure 24* shows the Send Queue with the "MS LoopBack Driver" interface selected as the target for broadcast.

Figure 24 - Select the Loopback Interface



Select Network Interface:

```
0.0.0.0
169.254.228.65
192.168.1.19
192.168.1.18
0.0.0.0
```

Device Information: MS LoopBack Driver

Send only selected packets

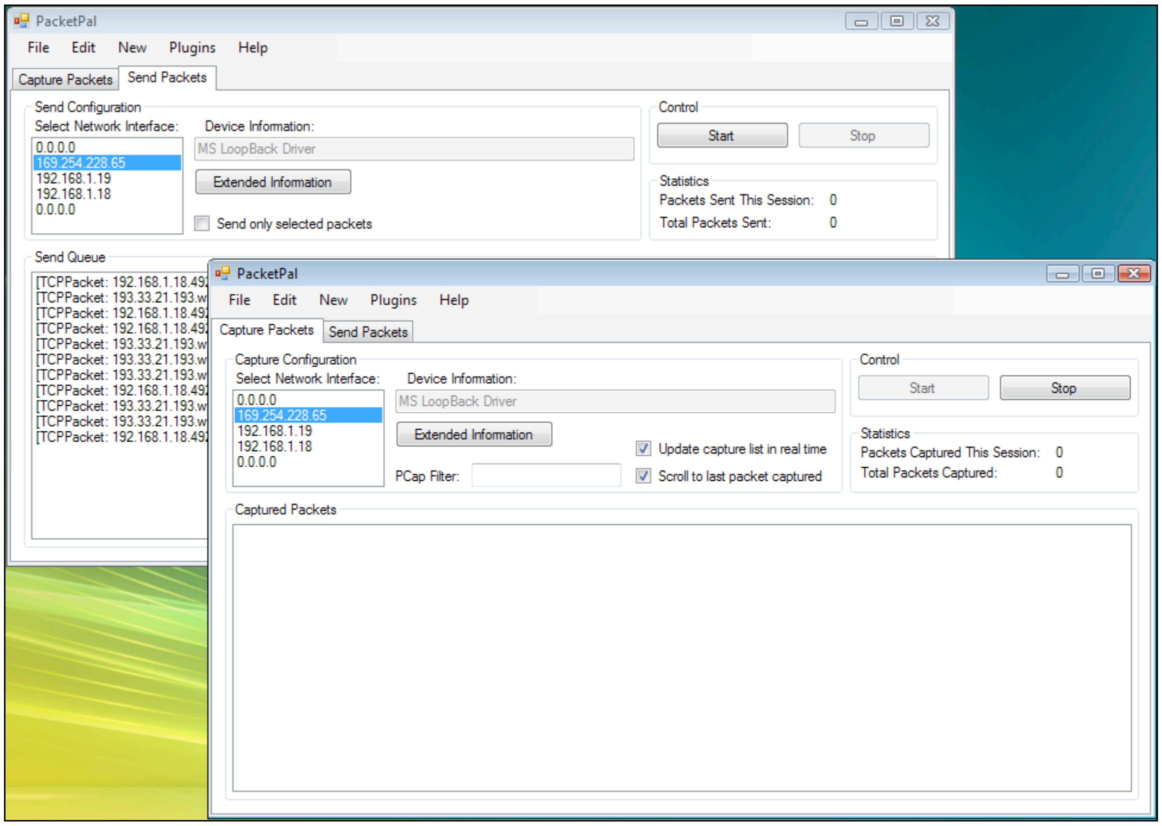
Packets Sent This Session: 0

Total Packets Sent: 0

```
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www syn[0xe27dde09] l=32,0]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27dde0a] syn[0x63f3b873] l=32,0]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] l=20,0]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] psh l=20,528]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,6]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3c3dc] l=20,0]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] psh l=20,650]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3cc1a] l=20,0]
```

Now, an additional instance of Packet Pal must be opened. Select the MS LoopBack Driver as the target device for capturing. Press "Start" to begin capturing packets on the loopback interface. *Figure 25* shows a second instance of Packet Pal listening on the loopback interface with the original instance of Packet Pal in the background.

Figure 25 - Capturing with another instance of Packet Pal



```
Send Configuration
Select Network Interface:      Device Information:
0.0.0.0                        MS LoopBack Driver
169.254.228.65
192.168.1.19
192.168.1.18
0.0.0.0
[ ] Send only selected packets

Statistics
Packets Sent This Session:   0
Total Packets Sent:          0

Send Queue
[TCPPacket: 192.168.1.18.49
[TCPPacket: 193.33.21.193.w
[TCPPacket: 192.168.1.18.49
[TCPPacket: 192.168.1.18.49
[TCPPacket: 193.33.21.193.w
[TCPPacket: 193.33.21.193.w
[TCPPacket: 193.33.21.193.w
[TCPPacket: 192.168.1.18.49
[TCPPacket: 193.33.21.193.w
[TCPPacket: 193.33.21.193.w
[TCPPacket: 192.168.1.18.49
```

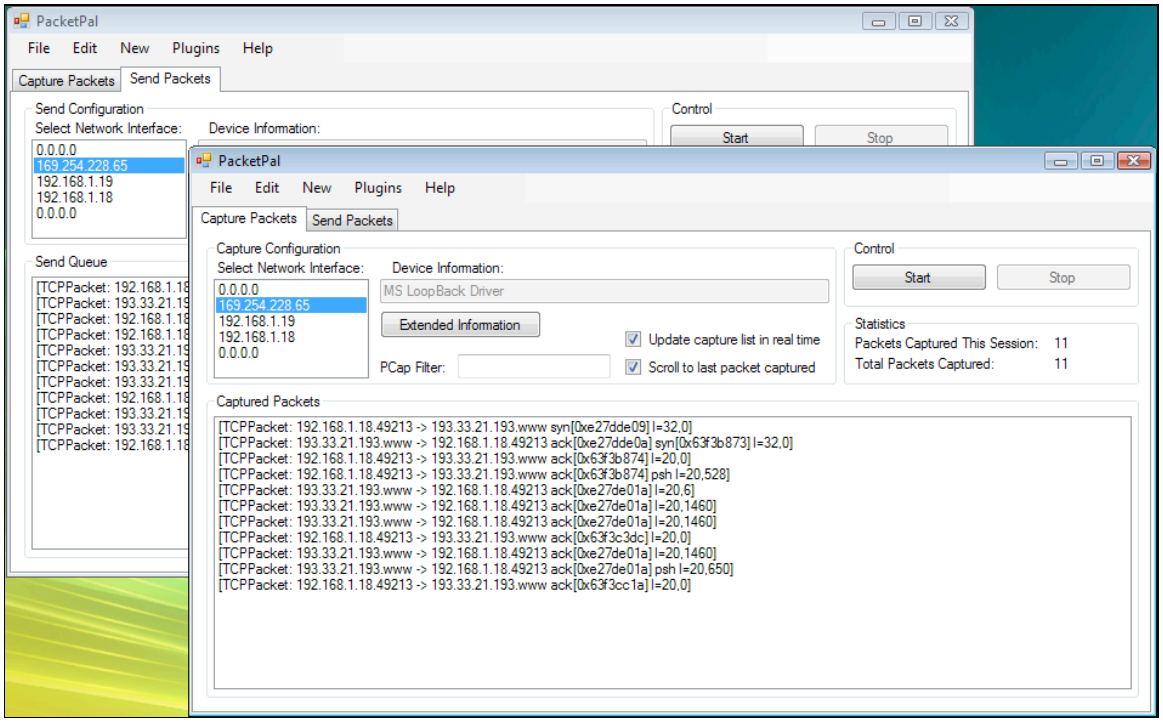
```
Capture Configuration
Select Network Interface:      Device Information:
0.0.0.0                        MS LoopBack Driver
169.254.228.65
192.168.1.19                   [✓] Update capture list in real time
192.168.1.18
0.0.0.0                        PCap Filter:       [✓] Scroll to last packet captured

Statistics
Packets Captured This Session:   0
Total Packets Captured:          0

Captured Packets
```

Return to the first instance of Packet Pal, with the previously captured packets already in the "Send Queue" and the loopback interface selected. Press "Start" to begin sending the packets. Immediately, the packets should show up in the newer instance of Packet Pal, which is already listening on the loopback interface. *Figure 26* shows the packets captured in the second instance of Packet Pal, sent by the first instance.

Figure 26 - Capture Results



Select Network Interface:

```
0.0.0.0
169.254.228.65
192.168.1.19
192.168.1.18
0.0.0.0
```

Device Information: MS LoopBack Driver

☑ Update capture list in real time

☑ Scroll to last packet captured

Packets Captured This Session: 11

Total Packets Captured: 11

Send Queue:

```
[TCPPacket: 192.168.1.18
[TCPPacket: 193.33.21.19
[TCPPacket: 192.168.1.18
[TCPPacket: 192.168.1.18
[TCPPacket: 193.33.21.19
[TCPPacket: 193.33.21.19
[TCPPacket: 193.33.21.19
[TCPPacket: 192.168.1.18
[TCPPacket: 193.33.21.19
[TCPPacket: 193.33.21.19
[TCPPacket: 192.168.1.18
```

Captured Packets:

```
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www syn[0xe27dde09] l=32,0]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27dde0a] syn[0x63f3b873] l=32,0]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] l=20,0]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3b874] psh l=20,528]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,6]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3c3dc] l=20,0]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] l=20,1460]
[TCPPacket: 193.33.21.193.www -> 192.168.1.18.49213 ack[0xe27de01a] psh l=20,650]
[TCPPacket: 192.168.1.18.49213 -> 193.33.21.193.www ack[0x63f3cc1a] l=20,0]
```

This concludes the demonstration using Packet Pal with the Microsoft LoopBack interface.

Further Research

There are many potential research opportunities relating to Packet Pal, due to its extensible design. Packet Pal can be easily modified and extended to allow continuing research in several networking-related research fields. Some potential projects that arose throughout the course of this project are listed below:

- ASCII (or other plain text) and hexadecimal side-by-side editing display. This was attempted during one of the early implementations of Packet Pal, but was inefficient and significantly slowed the editing interface.
- Generation of packets based on Snort rules. Snort is a powerful intrusion detection system with advanced rule sets for detecting intrusions. The generation of traffic based on these rules becomes a problem similar to generating a test program based on a Backus-Naur Form (BNF) language definition. This could be implemented as a plug-in.
- Reconstruction of TCP communication streams. This feature is useful to see the entire communication between two parties, as TCP communication protocols often use plain text. This could be implemented as a plug-in.
- Integration of the libwireshark library. The libwireshark library is the core

of Wireshark and has many advanced features, in comparison to SharpPcap, and offers hundreds of pre-defined parsers (called "dissectors" in Wireshark), as opposed to the very small list of predefined parsers in SharpPcap. Because of its large and complicated list of features, implementing Packet Pal on top of libwireshark presents a significant challenge, but could yield tremendous results [2].

- Dynamic widget generation with editable elements for the display of packet details. This could be very useful for creating new editing interfaces, especially if an interaction with libwireshark is achieved, as creating new interfaces for hundreds of protocols would be time consuming.
- Implementation of new, even experimental, protocols using the application programming interfaces provided in Packet Pal. The protocols could then be used and tested with Packet Pal.

The above list does not include smaller projects, such as modifications to existing interfaces, creation of new interfaces for specific fields, and creation of plug-ins for specific fields.

Conclusion

The intention of this project was to develop a tool that would be useful in an educational environment and provide an extensible platform for further development. Packet Pal allows for the capture of network traffic through the libpcap library, allowing for saved data compatibility with other libpcap-based products, such as Wireshark. The Packet Editors included in Packet Pal allow for the intuitive modification of packets, using a layout similar to the educational diagrams used in modern networking books. The ability to re-broadcast captured, modified, and generated packets, using a **GUI** allows the user to easily simulate traffic and aids in demonstrations. The capture and broadcast interfaces in Packet Pal are designed to parallel popular industry standard tools, therefore making it easier to transition between these tools. Users already familiar with these tools should be able to use Packet Pal intuitively. Because Packet Pal is open-source, there is increased potential for an active community of developers to continue research and develop needed extensions. Packet Pal satisfies the implementation goals of this project and maintains an educational focus, while remaining desirable for other industries.

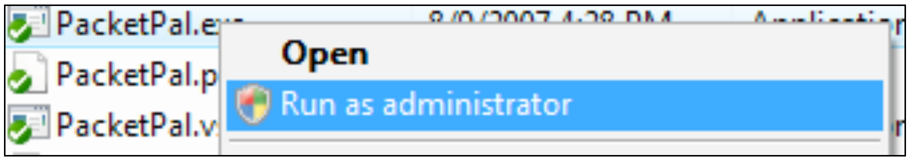
The plug-in **API** and Packet Editor **API** provide a solid but malleable platform for further development and research. Packet Pal was designed using an object-oriented paradigm, which allows for the project to be constructed as individual modules, which is reflected in the **APIs** available for further development. This practice encourages code reuse throughout the project, and also makes understanding, modification, and upgrading the code relatively easy.

As shown, there are other open-source and commercially developed tools that, in conjunction with other tools, are able to potentially reach the same goals as Packet Pal. Packet Pal, however, brings all of these important features into one package, at no charge to the end user.

Appendix A: Packet Pal Help

Problem: In Windows Vista, Packet Pal says that no available interfaces were found, but WinPcap is installed properly.

Solution: Packet Pal will need to be run with administrative privileges in order to access the network hardware under Windows Vista. To do this, right-click on the Packet Pal executable, and select "Run as administrator".



When the "User Account Control" dialog is displayed, select "Allow". Packet Pal should now have the needed access to the network hardware

Problem: There is no loopback interface available for use in Packet Pal.

Solution: To solve this problem, the Microsoft LoopBack Driver must be installed on the host Windows machine in which Packet Pal is running. The steps below will help with the driver installation.

This solution is derived from Microsoft's Help and Support article "How to install the Microsoft Loopback adapter in Windows XP". The original article can be viewed here: http://support.microsoft.com/kb/839013 [11].

Steps for Windows Vista 32

- Open the Control Panel.
- Click "Switch to Classic View" on the left side pane if it is not already in Classic View.
- Open the "Add Hardware" application.
- When Windows asks for permission to continue, select "Continue."
- In the Add Hardware Wizard welcome page, click "Next."
- Select the option to "Install the hardware that I manually select from a list (Advanced)" and click "Next."
- Scroll down the list of "Common hardware types" and select "Network adapters", then click "Next."
- In the "Manufacturer" list, select "Microsoft."
- In the "Network Adapter" list, select "Microsoft Loopback Adapter" and click "Next."
- Click "Next" again, and the driver will install.
  Click "Finish" after the installation is complete to close the Add Hardware

Wizard.

Steps for Windows XP

- Open the Control Panel.
- Click “Switch to Classic View” on the left side pane if it is not already in Classic View.
- Open “Add Hardware.”
- In the Add Hardware Wizard welcome page, click “Next.”
- Windows will search for a new device for a few seconds. When it doesn’t find anything, it will ask if the hardware is already connected. Select “Yes, I have already connected the hardware” and click “Next.”
- You will then be presented with a list of Installed Hardware. Scroll all the way to the bottom and select “Add a new hardware device” and click “Next.”
- On the next screen, select “Install the hardware that I manually select from a list (Advanced)” and click “Next.”
- Scroll down the “Common Hardware Types” list and select “Network Adapters” then click “Next.”
- In the “Manufacturer” list, select “Microsoft.”
- In the “Network Adapter” list, select “Microsoft Loopback Adapter” and click “Next.”
- Click “Next” again to install the driver.
- Click “Finish” after the installation is complete to close the Add Hardware Wizard.

After the installation is complete, start up Packet Pal, or restart if it was running, and there will be a new device in the Network Interface list with a non-routable IP address. If this device is selected, the “Device Information” will read “MS LoopBack Driver”. To send and receive packets from the loopback address, use this network interface.

Remember that any traffic generated on the loopback address does not ever reach and actual network interface, so it will stay not leave the host machine. See the *Demonstration Using the Microsoft LoopBack Interface* section for more information on how to use the Microsoft LoopBack Driver.

Appendix B: References

1. Berghel, Dr. Hal. “Packet Pal Primer.”
   http://www.berghel.net/resources/packetpal/index.php

2. Combs, Gerald. “Wireshark.”
   http://www.wireshark.org

3. Comer, Douglas E. Internetworking with TCP/IP. Upper Saddle River, NJ: Prentice Hall, 2000.

4. Fydor. “Top 100 Network Security Tools.”
   http://sectools.org

5. Gal, Tamir. “SharpPcap – A packet capture framework for .NET.” ©2005 TamirGal.com.
   http://www.tamirgal.com

6. Jacobson, Van, Craig Leres, and Steven McCanne. “Tcpdump / libpcap.”
   http://www.tcpdump.org

7. Priddy, Brent. “Cheops-ng.”
   http://cheops-ng.sourceforge.net

8. Stevens, W. Richard. TCP/IP Illustrated, Volume 1. Indianapolis, IN: Addison-Wesley, 1994.

9. Turner, Aaron. “Tcpreplay: Pcap editing and replay tools for *NIX.”
   http://tcpreplay.synfin.net

10. “Ethereal: A Network Protocol Analyzer.” Ethereal, Inc.
    http://www.ethereal.com

11. “How to install the Microsoft Loopback adapter in Windows XP.” Microsoft Help and Support. ©2007 Microsoft Corporation.
    http://support.microsoft.com/kb/839013

12. “Jpcap Network Packet Capture Library.”
    http://jpcap.sourceforge.net

13. “Maven Security Consulting, Inc. – Achilles.” ©2001-2006 Maven Security Consulting, Inc.
    http://www.mavensecurity.com/achilles

14. “The Metasploit Project.” ©2003-2007 Metasploit, LLC.

http://www.metasploit.org

15. "NetScan Tools Pro." ©2007 Northwest Performance Software, Inc.
http://www.netscantools.com

16. "Network Analysis Tools & Security Software." ©1998-2007 TamoSoft.
http://www.tamos.com

17. "NSAuditor." ©2007 NsaSoft.
http://www.nsauditor.com

18. "Snort –The de facto standard for intrusion detection/prevention." ©2007 Sourcefire, Inc.
http://www.snort.org

19. "TCP/IP Fundamentals for Microsoft Windows." ©2007 Microsoft Corporation.
http://technet.microsoft.com/en-us/library/bb726983.aspx

20. "WinPcap, The Packet Capture and Monitoring Library for Windows."
http://www.winpcap.org

Appendix C: Glossary

Achilles – A network security assessment tool that allows the user to intercept, modify, and log network traffic [13].

API – Application Programming Interface. An API is an interface used by an application or library to provide support for inter-program communication.

Cheops – A tool for mapping and monitoring networks [7].

Ethereal – The predecessor of Wireshark. Ethereal is an open source network packet capture utility and network protocol analyzer [10] [2].

GUI – Graphical User Interface. The visual aspect of a program used to facilitate interactions between the user and the program.

Hunt – A tool that allows the user to intrude on, interrupt, and reset network connections [4].

Metasploit – An open source framework for exploiting network and other vulnerabilities, designed for penetration testers [14].

OSI – An acronym for the Open Systems Interconnection initiative. OSI was an effort to standardize networking by the International Standards Organization (ISO) that began in 1982.

TCP/IP – A collection of network communication protocols, also known as the "Internet Protocol Suite."

Tcpdump – An open source command line application that uses libpcap to capture network traffic [6].

Appendix D: Code for Packet Pal

The source code for Packet Pal can be found on the compact disk included in a sleeve the back cover of this document.