General Programming » Internet / Network » Utilities

C++, Windows, Visual Studio, MFC, Dev

Posted:	**1 Jul 2007**
Updated:	**22 Jul 2007**
Views:	**68,945**
Bookmarked:	**52 times**

Creating Your Own Custom Wireshark Dissector

By **KenThompson**

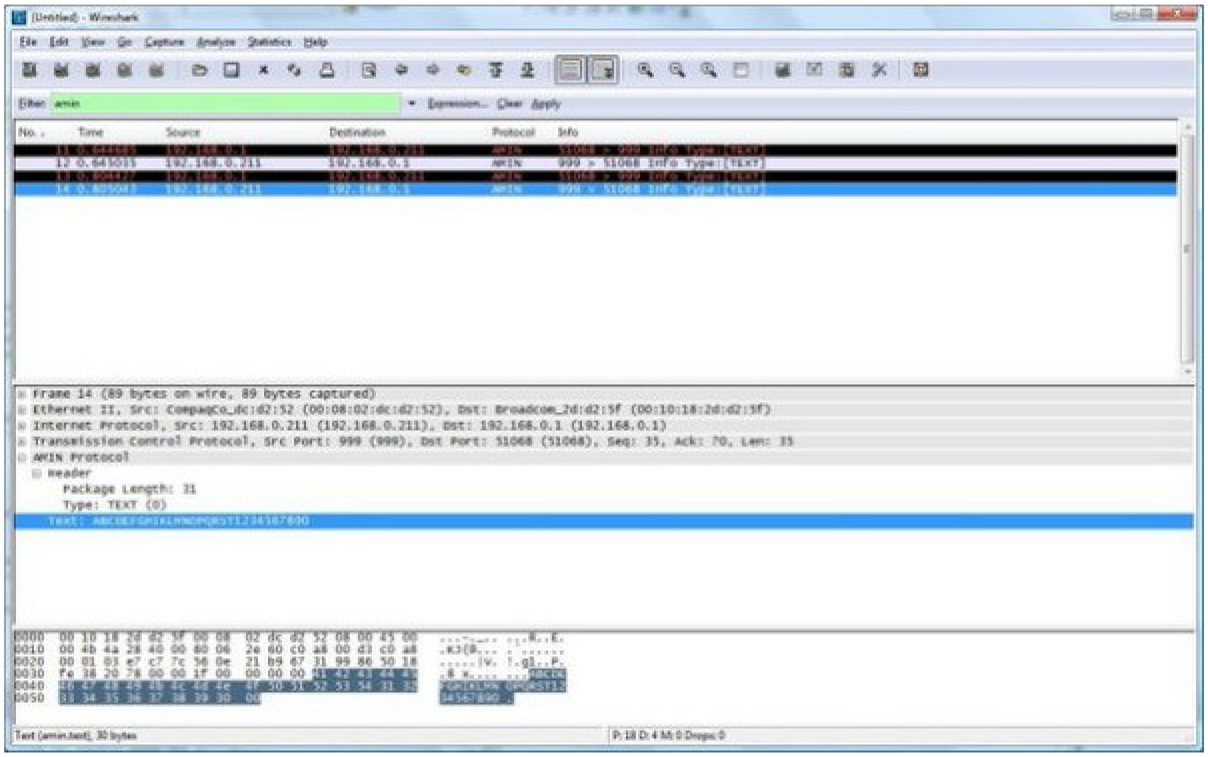
This article describes how to create a Wireshark dissector as well as how to setup the Wireshark build environment.

17 votes for this Article.

Popularity: 5.37 Rating: **4.37** out of 5

1 2 3 4 5

Download source - 9.32 KB


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

```
No. .  Time        Source          Destination     Protocol  Info
   11 0.644685    192.168.0.1     192.168.0.211   AMIN      51068 > 999 Info Type:[TEXT]
   12 0.645035    192.168.0.211   192.168.0.1     AMIN      999 > 51068 Info Type:[TEXT]
   13 0.804427    192.168.0.1     192.168.0.211   AMIN      51068 > 999 Info Type:[TEXT]
   14 0.805043    192.168.0.211   192.168.0.1     AMIN      999 > 51068 Info Type:[TEXT]
```

```
⊞ Frame 14 (89 bytes on wire, 89 bytes captured)
⊞ Ethernet II, Src: CompaqCo_dc:d2:52 (00:08:02:dc:d2:52), Dst: Broadcom_2d:d2:5f (00:10:18:2d:d2:5f)
⊞ Internet Protocol, Src: 192.168.0.211 (192.168.0.211), Dst: 192.168.0.1 (192.168.0.1)
⊞ Transmission Control Protocol, Src Port: 999 (999), Dst Port: 51068 (51068), Seq: 35, Ack: 70, Len: 35
⊟ AMIN Protocol
  ⊟ Header
      Package Length: 31
      Type: TEXT (0)
    Text: ABCDEFGHIJKLMNOPQRST1234567890
```

```
0000  00 10 18 2d d2 5f 00 08  02 dc d2 52 08 00 45 00   ...-._.. ...R..E.
0010  00 4b 4a 28 40 00 80 06  2e 60 c0 a8 00 d3 c0 a8   .KJ(@... .`......
0020  00 01 03 e7 c7 7c 36 0e  21 b9 67 31 99 86 50 18   .....|6. !.g1..P.
0030  fe 38 20 78 00 00 1f 00  00 00 00 41 42 43 44 45   .8 x.... ...ABCDE
0040  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 31   FGHIJKLM NOPQRST1
0050  32 33 34 35 36 37 38 39  30 00                     23456789 0.
```

1.0 Introduction

Wireshark is a powerful open source tool used to dissect Ethernet packets. Have you ever wondered what it takes to implement your own custom dissector? Furthermore, have you attempted to learn Wireshark's API and found it difficult to understand? This article will attempt to demystify the development of your very own protocol dissector. This article uses Amin Gholiha's "A Simple IOCP Server/Client class" [^] as a basis for dissection, thus producing the AMIN protocol.

1.1 Requirements

- This article expects the reader to be familiar with structured C, TCP/IP.
- The source code has been designed to compile on Windows. A Linux version of this article may be produced at another time.
- A basic knowledge of how to use Wireshark to capture packets.
- A C++ compiler. VS2005/VS2003/VC++6/VS2005EE.

2.0 Configure Wireshark Build Environment (Win32)

The Wireshark developer's guide [^] features a section on setting up the Win32 environment, which I found to be invaluable. This section will paraphrase much of the information found there.

Step 1. C Compiler

If you do not have VS2005/VS2003 etc., you will need to download and install "Visual C++ 2005 Express Edition"[^].

Step 2. Platform SDK

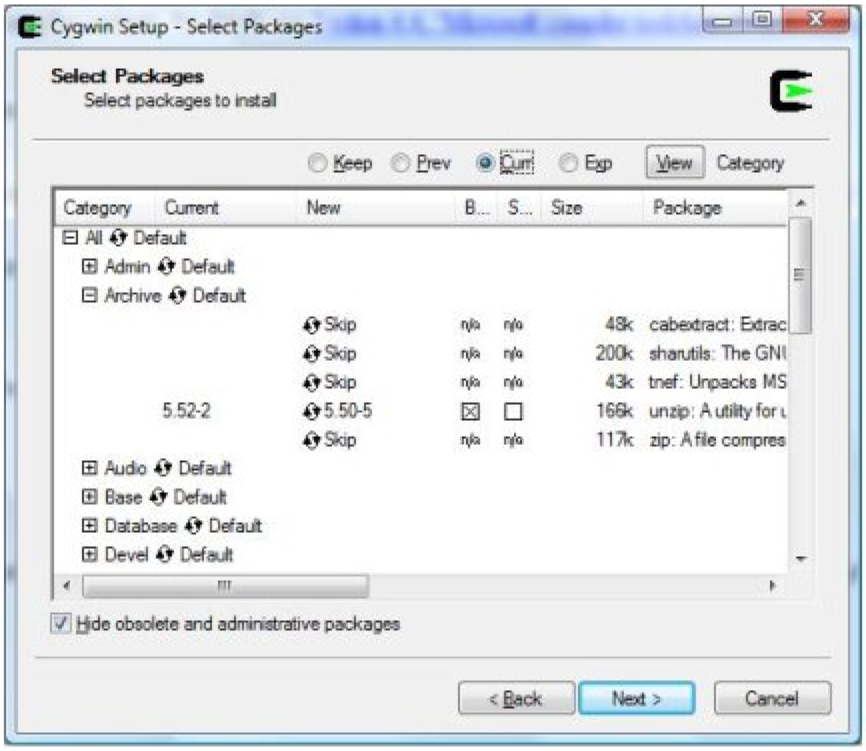
You must download and install the Platform SDK Server 2003 R2[^].

Step 3. Install Cygwin

This guide will not go into great details about the Cygwin package. In short, it allows Wireshark to be compiled on Windows and Linux – which is quite a feat.
Download the Cygwin installer and start it.

At the "Select Packages" page, you will need to select some additional packages which are not installed by default. Navigate to the required Category/Package row, and click on the "Skip" item in the "New" column so it shows a version number for:

- Archive/unzip
- Devel/bison
- Devel/flex
- Interpreters/perl
- Utils/patch
- Web/wget



Cygwin Setup - Select Packages

Select Packages
Select packages to install

( ) Keep ( ) Prev (●) Cur ( ) Exp View Category

Category	Current	New	B...	S...	Size	Package
⊟ All Default						
⊞ Admin Default						
⊟ Archive Default						
		Skip	n/a	n/a	48k	cabextract: Extrac
		Skip	n/a	n/a	200k	sharutils: The GNU
		Skip	n/a	n/a	43k	tnef: Unpacks MS
	5.52-2	5.50-5	☒	☐	166k	unzip: A utility for u
		Skip	n/a	n/a	117k	zip: A file compres
⊞ Audio Default						
⊞ Base Default						
⊞ Database Default						
⊞ Devel Default						

[✓] Hide obsolete and administrative packages

< Back Next > Cancel

After clicking the Next button several times, the setup will then download and install the selected packages (this may take a while).

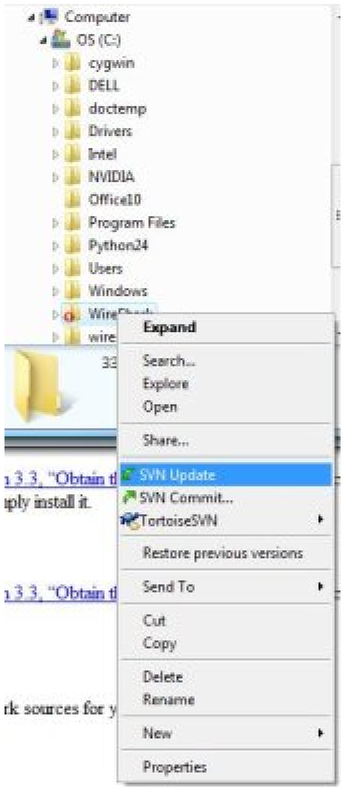
Step 4. Install Python

Get the Python 2.4 installer and install Python into the default location. Note: Python 2.5 doesn't work out of the box, so avoid it.

Step 5. Install Subversion Client

Subversion is not required to build Wireshark, but it is required to follow this article. Subversion allows you to grab the latest source to work with.

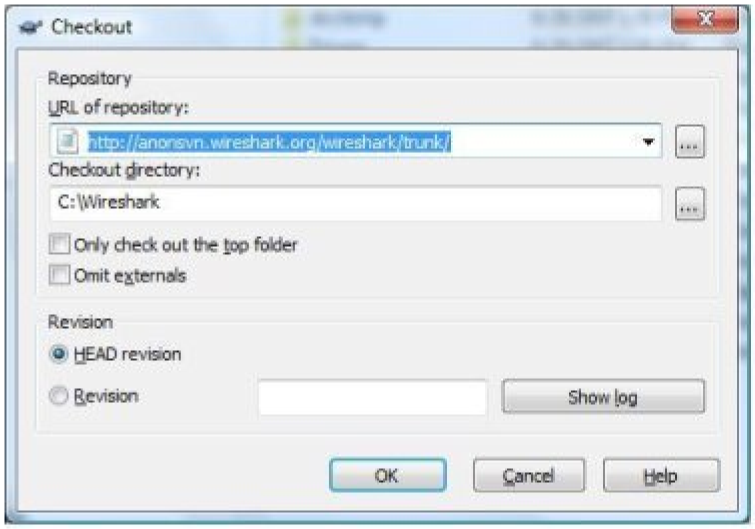
My personal choice for a subversion client is TortoiseSVN[^]. Download and install TortoiseSVN[^]. You may need to reboot after the installation to enable the context menu options. This is a pretty nifty feature that lets you right-click on folders in the "explorer" view and grab a source tree.



Step 6. Get the Latest Wireshark Source

You may be wondering at this point why you are getting the entire source tree just to make a dissector. The source tree contains the "*plugins*" directories that contain examples and a place to build your dissector.

a. Right-click on the *C:\* drive in Windows Explorer.
b. In the upcoming context menu, select "SVN checkout.." and then set:
c. The URL of the repository to *http://anonsvn.wireshark.org/wireshark/trunk/*.
d. The checkout directory to *c:\wireshark*.



URL of repository: http://anonsvn.wireshark.org/wireshark/trunk/
Checkout directory: C:\Wireshark

e. Select OK. You may be prompted to create the Wireshark directory. Select *Yes*.
f. TortoiseSVN starts downloading the source.

g. If the download fails, you may be behind a restrictive firewall. In this case, you better find an alternate way to get the source tree. (You are welcome to email me a request.)

Step 7. Edit the config.nmake

Open *C:\Wireshark\config.nmake* using Notepad or your favorite text editor. The following sections must be updated:

- **VERSION_EXTRA**: Give Wireshark your "private" version info. E.g.: -myprotocol123. Mine reads: SVN-AMINPROTOCOL. You are welcome to put whatever you wish here.
- **PROGRAM_FILES**: Where your programs resides. Usually, just keep the default.
- **MSCV_VARIANT**: This part is *critical*. If your compiler isn't uncommented by default, you must **comment out** the current selection and **uncomment** your own. For example:

```
# "Microsoft Visual Studio 6.0" - RECOMMENDED
# Visual C++ 6.0, _MSC_VER 1200, msvcrt.dll (version 6)
#MSVC_VARIANT=MSVC6

# "Microsoft Visual Studio .NET (2002)" - WORKS
# Visual C++ 7.0, _MSC_VER 1300, msvcr70.dll
#MSVC_VARIANT=MSVC2002

# "Microsoft .Net Framework SDK Version 1.0" - WORKS
# needs additional Platform SDK installation
# Visual C++ 7.0, _MSC_VER 1300, msvcr70.dll
#MSVC_VARIANT=DOTNET10

# "Microsoft Visual Studio .NET 2003" - WORKS
# Visual C++ 7.1, _MSC_VER 1310, msvcr71.dll
#MSVC_VARIANT=MSVC2003

# "Microsoft .Net Framework SDK Version 1.1" - WORKS
# needs additional Platform SDK installation
# Visual C++ 7.1, _MSC_VER 1310, msvcr71.dll
#MSVC_VARIANT=DOTNET11

# "Microsoft Visual Studio 2005" - WORKS
# Visual C++ 8.0, _MSC_VER 1400, msvcr80.dll
MSVC_VARIANT=MSVC2005    <------ This is my compiler. All others are commented out.

# "Microsoft Visual C++ 2005 Express Edition" - WORKS
# needs additional Platform SDK installation
# Visual C++ 8.0, _MSC_VER 1400, msvcr80.dll
#MSVC_VARIANT=MSVC2005EE

# "Microsoft .Net Framework 2.0 SDK" - WORKS
# needs additional Platform SDK installation
# Visual C++ 8.0, _MSC_VER 1400, msvcr80.dll
#MSVC_VARIANT=DOTNET20
```

Step 8. Prepare cmd.exe

a. Start *cmd.exe* (Start > Run > "cmd")
b. Call "*C:\Program Files\Microsoft Platform SDK for Windows Server 2003 R2\SetEnv.Cmd*"
c. Call "*C:\Program Files\Microsoft Visual Studio 8\VC\bin\vcvars32.bat*"
d. *cd c:\wireshark*

Your paths may be slightly different depending on the compiler you use. In the example zip file, you will find a *step1.bat*, *step2.bat*, and a *step3.bat*. I made these batch files to simplify this step. I know there are more fancy ways of constructing the batch files, so please feel free to submit your own.

Step 9. Verify Tools are Installed

From the *C:\WireShark* directory, execute:

```
C:\wireshark> Nmake -f Makefile.nmake verify_tools
```

Your output should look like:

```
cl: /cygdrive/c/Programme/Microsoft Visual Studio 8/VC/BIN/cl
link: /cygdrive/c/Programme/Microsoft Visual Studio 8/VC/BIN/link
nmake: /cygdrive/c/Programme/Microsoft Visual Studio 8/VC/BIN/nmake
bash: /usr/bin/bash
bison: /usr/bin/bison
flex: /usr/bin/flex
env: /usr/bin/env
grep: /usr/bin/grep
/usr/bin/find: /usr/bin/find
perl: /usr/bin/perl
env: /usr/bin/env
C:/python24/python.exe: /cygdrive/c/python24/python.exe
sed: /usr/bin/sed
unzip: /usr/bin/unzip
wget: /usr/bin/wget
```

If something is missing, you may have to repeat the Cygwin install.

Step 10. Install Libraries

If you have closed your *cmd.exe*, you will have to reopen it and execute *Step 8*. You can use the *step1*, *step2*, *step3* batch files to simplify the process. From *C:\Wireshark*, execute:

```
nmake -f Makefile.nmake setup (This step may take a little while to complete.)

nmake -f Makefile.nmake distclean
```

Step 11. Build Wireshark

If you have closed your *cmd.exe*, you will have to reopen it and execute *Step 8*. You can use the *step1*, *step2*, *step3* batch files to simplify the process.

```
nmake -f Makefile.nmake all  (this step will take quite a while to complete)
```

You're done; provided the above steps worked!

Step 12. (Optional) Create a Wireshark Installer

I like the idea of distributing my version to friends. You can easily create an installer by doing the following:

a. Download and install NSIS [^].

b. Download *vcredist_x86.exe* [^] and place it in the *c:\wireshark-win32\libs* directory. You may already have it in the directory, so check first.
c. Using the command line, prepare it using Step 8 or the *Step1/2/3* batch files provided, and from *C:\Wireshark*, execute:

```
nmake -f Makefile.nmake packaging
```

d. You should now have a *c:\wireshark\packaging\nsis\wireshark-setup.exe*. Try it out!

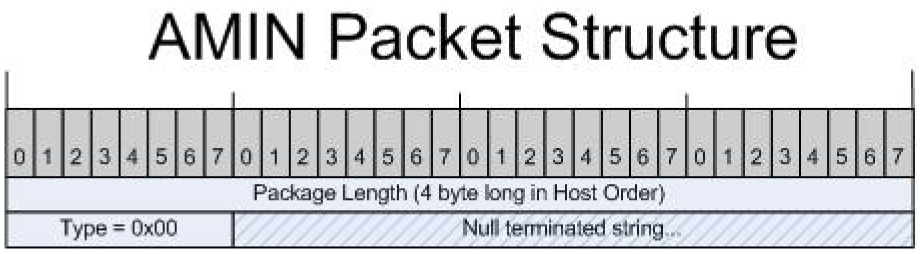
3.0 Execute the Compiled Wireshark

To launch Wireshark, you simply can run *C:\wireshark\wireshark-gtk2\wireshark.exe* and check if it starts. I found that I needed to download and install Wireshark before my build worked. This is probably due to not having WinPcap installed prior. You can download and install Wireshark from here.

Note: If you are using Visual Studio 2005, you will have to use your build of Wireshark to test the dissector. For some reason, compiling dissector plug-ins with Visual Studio 2005 prevents their use in the mainstream release of Wireshark. If you are using a different compiler, you may not have to use the compiled version of Wireshark.

4.0 Understanding the AMIN Protocol

Amin was nice enough to code up an excellent IOCP client/server example. His code uses a very simple protocol to transmit data via TCP. This article is designed for the beginner, so we will only be dissecting the text messages sent from the client/server. The file transfer messages are a tad more complex, so we will ignore those for now.



4.1 Package Length Prefix

All of Amin's packets are prefixed with a four (4) byte long value which indicates the size of the package. This value is presented in network order over the wire, or, MSB. Network order means that the bytes are ordered in such a way that the most significant bytes are first. For example, let's say your length is 12 decimal. This would be 0x0000000c in hex (if you are storing the value in a long). If you read this value off the "wire", it would appear as *0c 00 00 00;* this is "Network" order. You may be wondering why the bytes are reversed. If you think about it as an array of bytes, instead of a long, each byte is written in order. In this case, byte[0] is 0x0c, so it's written first. This is an important concept to understand.

When we use Wireshark to dissect the packet, it is important to understand that 2 and 4 byte integer values are always written in "Network" order, or LSB. In other protocols, the order may be Host – which would be LSB.

Note: The length accounts for the bytes to follow, meaning that the four (4) bytes which represent the length value are not included. For example, if you sent a packet that has 12 characters of text, the length would be 12+1 = 13 (the one byte is the type), as opposed to 12+1+4 = 17.

4.2 Type

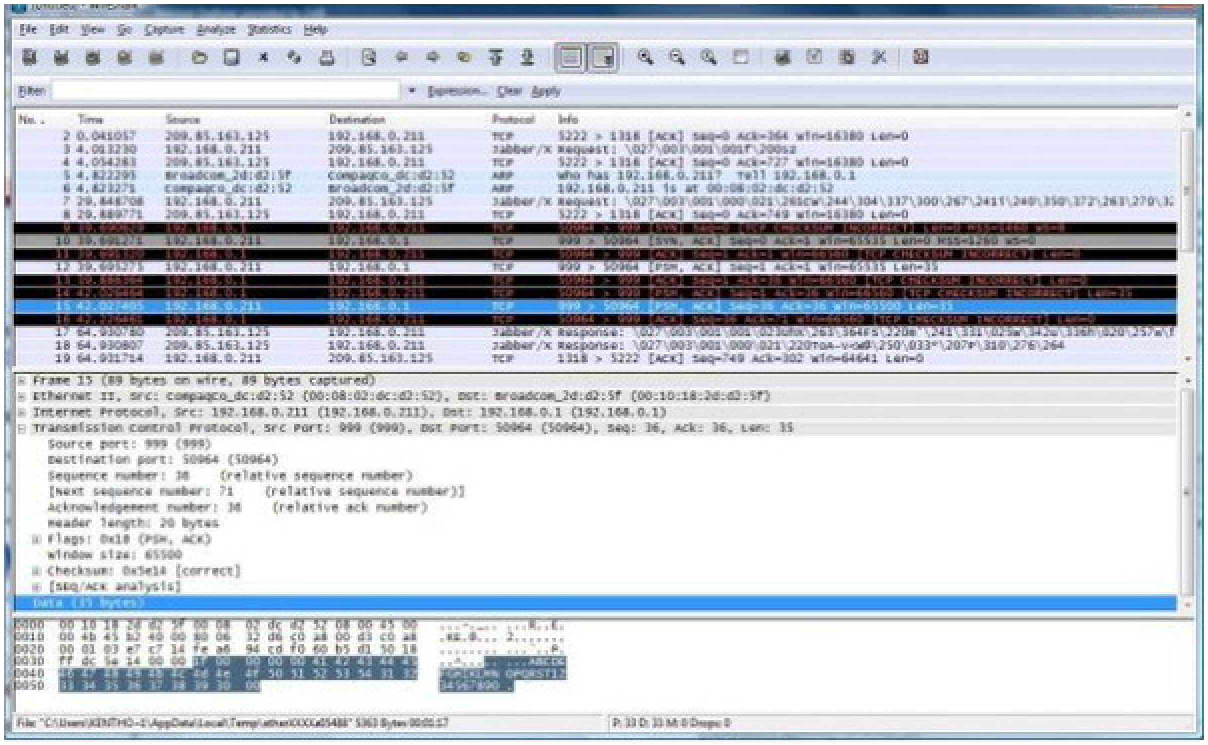
Amin follows the length prefix with a byte that indicates the "type" of the packet he is delivering. We will only be dealing with type 0x00, which is text. Note: a single byte is not subject to byte order because it's a single byte.

4.3 Null Terminated String

This field is simply a null terminated string which can be entered into Amin's MFC GUI. The default string is 'ABCDEFGHIJKLMNOPQRST123456789'.

5.0 Examining the AMIN protocol in Wireshark without a custom dissector

You can download the Simple IOCP Client/Server written by Amin here. Note: This article does not discuss the basic use of Wireshark. This article assumes you have used it before and understand how to use it in a basic sense. Here is what Wireshark looks like without a dissector for the AMIN protocol:


File Edit View Go Capture Analyze Statistics Help
Filter:
Expression... Clear Apply

```
No. .   Time       Source               Destination          Protocol   Info
     2 0.041057   209.85.163.125       192.168.0.211        TCP        5222 > 1318 [ACK] Seq=0 Ack=364 Win=16380 Len=0
     3 4.013230   192.168.0.211        209.85.163.125       Jabber/X   Request: \027\003\001\001f\200s2
     4 4.054283   209.85.163.125       192.168.0.211        TCP        5222 > 1318 [ACK] Seq=0 Ack=727 Win=16380 Len=0
     5 4.822295   Broadcom_2d:d2:5f    CompaqCo_dc:d2:52    ARP        Who has 192.168.0.211?  Tell 192.168.0.1
     6 4.823271   CompaqCo_dc:d2:52    Broadcom_2d:d2:5f    ARP        192.168.0.211 is at 00:08:02:dc:d2:52
     7 29.846708  192.168.0.211        209.85.163.125       Jabber/X   Request: \027\003\001\000\021\261Cw\244\304\337\300\267\2411\240\350\372\263\270\3:
     8 29.889771  209.85.163.125       192.168.0.211        TCP        5222 > 1318 [ACK] Seq=0 Ack=749 Win=16380 Len=0
     9 39.690829  192.168.0.1          192.168.0.211        TCP        50964 > 999 [SYN] Seq=0 [TCP CHECKSUM INCORRECT] Len=0 MSS=1460 WS=8
    10 39.691271  192.168.0.211        192.168.0.1          TCP        999 > 50964 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1260 WS=0
    11 39.691320  192.168.0.1          192.168.0.211        TCP        50964 > 999 [ACK] Seq=1 Ack=1 Win=66560 [TCP CHECKSUM INCORRECT] Len=0
    12 39.695275  192.168.0.211        192.168.0.1          TCP        999 > 50964 [PSH, ACK] Seq=1 Ack=1 Win=65535 Len=35
    13 39.886394  192.168.0.1          192.168.0.211        TCP        50964 > 999 [ACK] Seq=1 Ack=36 Win=66560 [TCP CHECKSUM INCORRECT] Len=0
    14 42.026464  192.168.0.1          192.168.0.211        TCP        50964 > 999 [PSH, ACK] Seq=1 Ack=36 Win=66560 [TCP CHECKSUM INCORRECT] Len=35
    15 42.027405  192.168.0.211        192.168.0.1          TCP        999 > 50964 [PSH, ACK] Seq=36 Ack=36 Win=65500 Len=35
    16 42.226465  192.168.0.1          192.168.0.211        TCP        50964 > 999 [ACK] Seq=36 Ack=71 Win=66560 [TCP CHECKSUM INCORRECT] Len=0
    17 64.930780  209.85.163.125       192.168.0.211        Jabber/X   Response: \027\003\001\001\023Uhx\263\364FS\220e'\241\331\025w\342u\336h\020\257w\f
    18 64.930807  209.85.163.125       192.168.0.211        Jabber/X   Response: \027\003\001\000\021\220ToA-v<w@\250\033*\207F\310\276\264
    19 64.931714  192.168.0.211        209.85.163.125       TCP        1318 > 5222 [ACK] Seq=749 Ack=302 Win=64641 Len=0

⊞ Frame 15 (89 bytes on wire, 89 bytes captured)
⊞ Ethernet II, Src: CompaqCo_dc:d2:52 (00:08:02:dc:d2:52), Dst: Broadcom_2d:d2:5f (00:10:18:2d:d2:5f)
⊞ Internet Protocol, Src: 192.168.0.211 (192.168.0.211), Dst: 192.168.0.1 (192.168.0.1)
⊟ Transmission Control Protocol, Src Port: 999 (999), Dst Port: 50964 (50964), Seq: 36, Ack: 36, Len: 35
    Source port: 999 (999)
    Destination port: 50964 (50964)
    Sequence number: 36    (relative sequence number)
    [Next sequence number: 71    (relative sequence number)]
    Acknowledgement number: 36    (relative ack number)
    Header length: 20 bytes
  ⊞ Flags: 0x18 (PSH, ACK)
    Window size: 65500
  ⊞ Checksum: 0x5e14 [correct]
  ⊞ [SEQ/ACK analysis]
  Data (35 bytes)

0000  00 10 18 2d d2 5f 00 08  02 dc d2 52 08 00 45 00   ...-._.. ...R..E.
0010  00 4b 45 b2 40 00 80 06  32 d6 c0 a8 00 d3 c0 a8   .KE.@... 2.......
0020  00 01 03 e7 c7 14 fe a6  94 cd f0 60 b5 d1 50 18   ........ ...`..P.
0030  ff dc 5e 14 00 00 1f 00  00 00 00 41 42 43 44 45   ..^..... ...ABCDE
0040  46 47 48 49 4a 4b 4c 4d  4e 4f 50 51 52 53 54 31   FGHIJKLM NOPQRST1
0050  32 33 34 35 36 37 38 39  00                        23456789 .
```

File: "C:\Users\KENTHO~1\AppData\Local\Temp\etherXXXXa05488" 5363 Bytes 00:01:17	P: 33 D: 33 M: 0 Drops: 0

Notice that we are simply given a field called "Data". Within the data portion, we can recognize our AMIN protocol based on the "1f 00 00 00 00" package length. The type follows, and after that is the "ABC.." ASCII.

6.0 Writing Your Own Dissector

The Wireshark source tree contains a directory called *Plugins*, which provides a reasonable amount of examples. However, it was impossible to find a really "simple" example to use. So, in combination with the H223 dissector, random examples from the internet, and the developer guide, I have prepared a simple example and placed it in the source zip file. The example can be found in the *AMIN* directory.

6.1 The make Files

In order to compile your own protocol, you must create a set of files to compile your dissector. The easiest thing to do is take the following files from the *AMIN* directory in the source zip file.

- *Makefile.am* - This is the UNIX/Linux makefile template
- *Makefile.common* - This contains the file names of this plug-in
- *Makefile.nmake* - This contains the Wireshark plug-in makefile for Windows
- *moduleinfo.h* - This contains the plug-in version info
- *moduleinfo.nmake* - This contains the DLL version info for Windows
- *packet-amin.c* - This is your dissector source
- *plugin.rc.in* - This contains the DLL resource template for Windows

Once you have copied the files into *C:\Wireshark\plugins\yourprotocolname*, you can begin editing them.

a. Change all occurrences of AMIN to your protocol name. Be sure to maintain capitalization. *Lines: 42, 43, 114*.
b. In *Makefile.common*, change all occurrences of AMIN to your protocol name. Be sure to maintain capitalization. *Line: 31.*
c. In *moduleinfo.h*, change all occurrences of AMIN to your protocol name. Be sure to maintain capitalization. *Line: 8*. On line 16, you can change the version to your own. This can be helpful if you plan to distribute your dissector.
d. In *moduleinfo.nmake*, change all occurrences of AMIN to your protocol name. Be sure to maintain capitalization on *Line 6. Lines 9, 10, 11, & 12* should be changed to reflect the version in *moduleinfo.h*, *Line: 16*.
e. In *packet-amin.c*, the first thing to do is change it so it reads *packet-yourprotocol.c*. All dissectors feature this convention, *packet-protocolname.c*. In the source file, you can simply do a "find and replace" for "amin". However, make sure that you replace the "AMIN" and "amin" references individually. In one case, you cannot submit the filter name in caps, and it will fail when you run Wireshark. Pay very close attention here.
f. You're done renaming things for now; now, on to the actual code!

7.0 Your Dissector Code

You can use a text editor of your choice to open *packet-yourprotocol.c*. Let's take it line by line:

```
#ifdef HAVE_CONFIG_H
# include "config.h"
#endif

#include <stdio.h>
#include <glib.h>
#include <epan/packet.h>
#include <string.h>
```

All dissectors use some standard headers. You need the *config.h*, *glib.h*, and *packet.h* for sure.

```
#define PROTO_TAG_AMIN    "AMIN"
```

I like to avoid hard coding constants when I can.

```
/* Wireshark ID of the AMIN protocol */
static int proto_amin = -1;
```

This value is very important to Wireshark. Wireshark uses this to identify our protocol.

```
/* These are the handles of our subdissectors */
static dissector_handle_t data_handle=NULL;
static dissector_handle_t amin_handle;
```

The dissector handle is what Wireshark uses to reference this dissector.

```
void dissect_amin(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree);
```

This is a forward declaration of our dissection function. We will pass this function to a registration function later on.

```
static int global_amin_port = 999;
```

This is the port that Wireshark will use to determine if the packet belongs to the AMIN protocol.

```
static const value_string packettypenames[] = {
    { 0, "TEXT" },
    { 1, "SOMETHING_ELSE" },
    { 0, NULL }
};
```

Here is where we add some optional text strings representing packet types. You can define many of these depending on the needs of your own protocol. It adds a level of detail that makes the dissector look well thought out.

```
static gint hf_amin = -1;
static gint hf_amin_header = -1;
static gint hf_amin_length = -1;
static gint hf_amin_type = -1;
static gint hf_amin_text = -1;


/* These are the ids of the subtrees that we may be creating */
static gint ett_amin = -1;
static gint ett_amin_header = -1;
static gint ett_amin_length = -1;
static gint ett_amin_type = -1;
static gint ett_amin_text = -1;
```

These allow us to attach IDs to the subcomponents of our protocol.

```
void proto_reg_handoff_amin(void)
{
    static gboolean initialized=FALSE;

    if (!initialized) {
        data_handle = find_dissector("data");
        amin_handle = create_dissector_handle(dissect_amin, proto_amin);
        dissector_add("tcp.port", global_amin_port, amin_handle);
    }
}
```

This function is called to register our protocol. Notice how the port and dissector handle are passed.

```
void proto_register_amin (void)
{
    /* A header field is something you can search/filter on.
    *
    * We create a structure to register our fields. It consists of an
    * array of hf_register_info structures, each of which are of the format
    * {&(field id), {name, abbrev, type, display, strings, bitmask, blurb, HFILL}}.
    */
    static hf_register_info hf[] = {
        { &hf_amin,
        { "Data", "amin.data", FT_NONE, BASE_NONE, NULL, 0x0,
        "AMIN PDU", HFILL }},
        { &hf_amin_header,
        { "Header", "amin.header", FT_NONE, BASE_NONE, NULL, 0x0,
         "AMIN Header", HFILL }},
        { &hf_amin_length,
        { "Package Length", "amin.len", FT_UINT32, BASE_DEC, NULL, 0x0,
        "Package Length", HFILL }},
        { &hf_amin_type,
        { "Type", "amin.type", FT_UINT8, BASE_DEC, VALS(packettypenames), 0x0,
         "Package Type", HFILL }},
        { &hf_amin_text,
        { "Text", "amin.text", FT_STRING, BASE_NONE, NULL, 0x0,
         "Text", HFILL }}
    };
```

The array above defines what elements we will be displaying. These declarations are simply a definition Wireshark uses to determine the data type, when we later dissect the packet.

```
    static gint *ett[] = {
        &ett_amin,
        &ett_amin_header,
        &ett_amin_length,
        &ett_amin_type,
        &ett_amin_text
    };
```

The array above simply attaches IDs to our definitions. Notice the 1:1 relationship of the `hf_` and `ett_` data types.

```
    proto_amin = proto_register_protocol ("AMIN Protocol", "AMIN", "amin");

    proto_register_field_array (proto_amin, hf, array_length (hf));
    proto_register_subtree_array (ett, array_length (ett));
    register_dissector("amin", dissect_amin, proto_amin);
```

The above code registers our protocol. In most of the examples I saw, they check to see if `proto_amin` is initialized already. However, a developer from the mailing list, "Jaap", emailed me saying it was not needed and confused the initalization process. The subsequent calls register our protocol and data handles.

```
static void
dissect_amin(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{
```

The dissect function is used to actually dissect and display the packet details.

First, some points are initialized to trees/items:

```
    proto_item *amin_item = NULL;
    proto_item *amin_sub_item = NULL;
    proto_tree *amin_tree = NULL;
    proto_tree *amin_header_tree = NULL;
    guint16 type = 0;
```

This next call checks to see if the INFO column is displaying our "AMIN" tag. If it's not, then we supply it.

```
    if (check_col(pinfo->cinfo, COL_PROTOCOL))
        col_set_str(pinfo->cinfo, COL_PROTOCOL, PROTO_TAG_AMIN);
    /* Clear out stuff in the info column */
    if(check_col(pinfo->cinfo,COL_INFO)){
        col_clear(pinfo->cinfo,COL_INFO);
    }

//Here we check to see if the INFO column is present. If it is we output
//which ports the packet came from and went to. Also, we indicate the type
//of packet.


    // This is not a good way of dissecting packets. The tvb length should
    // be sanity checked so we aren't going past the actual size.

    type = tvb_get_uint8( tvb, 4 ); // Get the type byte

    if (check_col(pinfo->cinfo, COL_INFO)) {
         col_add_fstr(pinfo->cinfo, COL_INFO, "%d > %d Info Type:[%s]",
         pinfo->srcport, pinfo->destport,
         val_to_str(type, packettypenames, "Unknown Type:0x%02x"));
    }
```

If there is a "tree" requested, we handle that request.

```
    if (tree) { /* we are being asked for details */
        guint32 offset = 0;
        guint32 length = 0;
```

This call adds our tree to the main dissection tree.

```
        amin_item = proto_tree_add_item(tree, proto_amin, tvb, 0, -1, FALSE);
        amin_tree = proto_item_add_subtree(amin_item, ett_amin);
        amin_header_tree = proto_item_add_subtree(amin_item, ett_amin);

        amin_sub_item = proto_tree_add_item( amin_tree, hf_amin_header,
                        tvb, offset, -1, FALSE );
```

Here, we add our own subtree, so we can have a "header" collapsible branch. Following the subtree, we copy out the length bytes to our data type. The order of the long value representing length is "network" order. If someone has a more appropriate `tvb_get` call to read the length value, it would be appreciated.

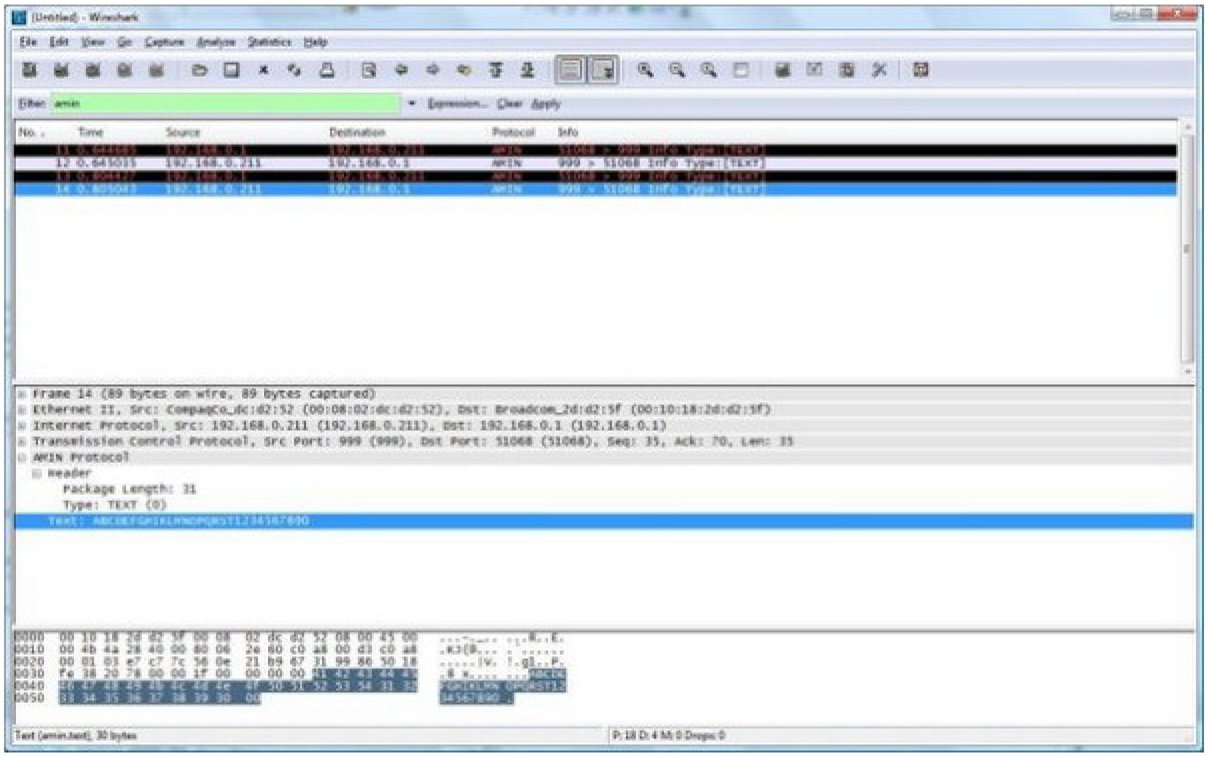
```
amin_header_tree = proto_item_add_subtree(amin_sub_item, ett_amin);

//We use tvb_memcpy to get our length value out (Host order)

tvb_memcpy(tvb, (guint8 *)&length, offset, 4);
proto_tree_add_uint(amin_header_tree, hf_amin_length, tvb, offset, 4, length);

//We increment the offset to get past the 4 bytes indicating length
offset+=4;

//Here we submit the type parameter to the tree.
proto_tree_add_item(amin_header_tree, hf_amin_type, tvb, offset, 1, FALSE);
type = tvb_get_guint8( tvb, offset ); // Get our type byte

offset+=1;

//If the type is TEXT, we add the text to the tree.

if( type == 0 )
{
    proto_tree_add_item( amin_tree, hf_amin_text, tvb, offset, length-1, FALSE );
}
```

7.1 Compiling packet-yourprotocol.c

If your command line window is still open, you can use that, or use *step1/2/3.bat* to arrive at the *c:\wireshark\plugins\yourprotocol* directory. If you are building my source code, you should be at *c:\wireshark\plugins\amin*. From the "prepared" command line, see *Step 8*. Execute:

```
nmake -f Makefile.nmake distclean
nmake -f Makefile.nmake all
```

If the build succeeds, you should have a *yourprotocol.dll*. Simply copy this file to your *C:\WireShark \wireshark-gtk2\plugins\0.99.7-YOUR-BUILD* directory. There should be a bunch of other dissector DLLs present there. You may need to copy it to your *c:\program files\wireshark\plugins\0.99.7-YOUR-BUILD* directory if you installed it there. The bottom line is to copy it in the appropriate directory based on how you are launching Wireshark. At this point, you should be able to launch Wireshark and dissect packets. A good test is to enter "yourprotocol" in the "filter" area of Wireshark and see if a green or red background forms. A green background, just like in the screenshots of this article, indicates that the dissector loaded correctly.


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

```
No. .  Time        Source           Destination      Protocol  Info
    11 0.644685    192.168.0.1      192.168.0.211    AMIN      51068 > 999 Info Type:[TEXT]
    12 0.645035    192.168.0.211    192.168.0.1      AMIN      999 > 51068 Info Type:[TEXT]
    13 0.804477    192.168.0.1      192.168.0.211    AMIN      51068 > 999 Info Type:[TEXT]
    14 0.805083    192.168.0.211    192.168.0.1      AMIN      999 > 51068 Info Type:[TEXT]
```

```
⊞ Frame 14 (89 bytes on wire, 89 bytes captured)
⊞ Ethernet II, Src: CompaqCo_dc:d2:52 (00:08:02:dc:d2:52), Dst: Broadcom_2d:d2:5f (00:10:18:2d:d2:5f)
⊞ Internet Protocol, Src: 192.168.0.211 (192.168.0.211), Dst: 192.168.0.1 (192.168.0.1)
⊞ Transmission Control Protocol, Src Port: 999 (999), Dst Port: 51068 (51068), Seq: 35, Ack: 70, Len: 35
⊟ AMIN Protocol
  ⊟ Header
      Package Length: 31
      Type: TEXT (0)
    Text: ABCDEFGHIKLMNOPQRST1234567890
```

```
0000  00 10 18 2d d2 5f 00 08  02 dc d2 52 08 00 45 00   ...-._.. ...R..E.
0010  00 4b 4a 28 40 00 80 06  2e 60 c0 a8 00 d3 c0 a8   .KJ(@... .`......
0020  00 01 03 e7 c7 7c 56 0e  21 b9 67 31 99 86 50 18   .....|V. !.g1..P.
0030  fe 38 20 78 00 00 1f 00  00 00 00 41 42 43 44 45   .8 x.... ...ABCDE
0040  46 47 48 49 4b 4c 4d 4e  4f 50 51 52 53 54 31 32   FGHIKLMN OPQRST12
0050  33 34 35 36 37 38 39 30  00                        34567890 .
```

```
Text (amin.text), 30 bytes                P: 18 D: 4 M: 0 Drops: 0
```

This is a screen capture of the AMIN protocol being dissected by Wireshark.

8.0 Tips

8.1 CodeBase

I found an immense amount of examples in a searchable form at CodeBase[^]. Without this site, it would have been a much more difficult task locating functions for dissection. Someone really should spend time documenting the Wireshark API in a more formal manner.

A coworker informed me that CodeBase is a pay service. I was able to get into CodeBase beta without having to authenticate. Use this link to do the same (use the search box to find undocumented methods).

8.2 Wireshark Filtering

After you've defined your data type map, you can use the '.' operator to limit the packets being displayed. For example, *amin.type==0* will only show packets where the type equals zero.

8.3 Other Examples

There are far more complex examples that can be found in the *plugins* directory. My suggestion is to check out the H223 dissector. Topics such as TCP segmentation are covered, as well as managing states across combinations of packets (i.e., watching a SIP session throughout multiple packets).

8.4 Capture Loopback Packets

If you wish to locally capture packets using Wireshark, i.e., 127.0.0.1, you must perform a few extra steps. I found this link which helps solve the problem. The required steps, per the site suggestion, are:

a. Install a loopback adapter. Windows 2003[^], Windows XP[^], Windows 2000[^]
b. Go to MS Loopback adapter properties, set:
    - IP 10.0.0.10
    - MASK 255.255.255.0
    - Adapter/additional/network address: 55-55-55-55-55-55
c. From the command line (*cmd.exe*):

    ```
    arp -s 10.0.0.10 55-55-55-55-55-55
    ```

    then:

    ```
    route add 10.0.0.10 10.0.0.10 mask 255.255.255.255
    ```

    You can then test the capture by executing:

    ```
    telnet 10.0.0.10
    ```

9.0 Conclusion

Once the complexity has been removed from designing dissectors, it's quite easy to make your very own protocol present in Wireshark. In addition, the guys at Wireshark greatly appreciate those who take the time to legally reverse engineer protocols for inclusion within the Wireshark distribution.

10.0 References

- "A Simple IOCP Server/Client Class", Amin Gholiha, The Code Project, 09/05/2005
- Wireshark Developer's Guide, 22223 For Wireshark 0.99.6
- H223 Example MX Telecom Ltd.
- Wireshark – Network Traffic Analyzer, Gerald Combs

11.0 History

- July 22, 2007 - Fixed a typo in *packet-amin.c*: tvb_get_uint8 -> tvb_get_guint8. Thanks Keith!
- July 9, 2007 - Added a CodeBase link that allows you to search without being a member of CodeBase. Not sure if this is legal, send me an email if it isn't.
- July 3, 2007 - Removed extra "nmake Makefile.nmake setup" in tools installation section.
- July 2, 2007 - Updated source code and article to reflect suggestions made by Jaap. Changes include:
    - Changed `static int intialized=FALSE` to `static gboolean`.
    - Removed `#include <gmodule.h>`, not needed.
    - Commented out `if(proto_amin == -1){}`.
    - Moved the `if(check_col(pinfo->cinfo, COLINFO)){` routine out of `if(tree)`.
    - Fixed comments on network versus host byte order.
    - Added section 8.4 on capturing loopback packets.
- July 1, 2007 - Spell checking and other grammatical corrections. Fixed an issue with

`packettypename[]` where `{0,NULL}` was required to avoid a seg fault. Thank you Ronnie.

- June 29, 2007 – First draft.

License

About the Author

KenThompson

Ken Thompson is a programmer working for EXACOM, Inc. Exacom is a telecommunications engineering firm specializing in public safety applications.

Ken graduated from University of New Hampshire with a B.S. in Electrical Engineering Technology.

Ken has been working in the telecommunications field since 1999. Since then he has been programming in C++ and with the advent of .NET an occasional C# application.

Email: kenthompson1+cp@gmail.com

Occupation:	Software Developer (Senior)
Company:	EXACOM, Inc.
Location:	United States

Discussions and Feedback

59 messages have been posted for this article. Visit **http://www.codeproject.com/KB/IP/custom_dissector.aspx** to post and view comments on this article, or click **here** to get a print view with messages.