Development of Network Protocol Resolver Based on Wireshark

Xiao-yu CUI* and Qing-guo SHEN
The Army Engineering University of PLA, Nanjing, China
*Corresponding author

Keywords: Wireshark, Network protocol, Protocol parser.

Abstract. Nowadays, the parsing of network protocol is widely used, we require a good and extensible protocol analysis tool. Wireshark is highly regarded by more and more people for its good system architecture and open source. This paper introduces the composition and function of Wireshark and studies the design and development method of protocol analysis parsers in Wireshark. It provides reference for the analysis and verification of network protocol and content detection of packets. In the meanwhile, it lays a foundation for the future design and development of more complex network application system.

Introduction

The parsing of network protocol refers to the process of parsing network data packet's protocol head and its load, so as to understand the behavior of network data packets in the process of generation and transmission. The software or device that contains the parser is called the protocol parser. Network protocol parse is widely used, such as monitoring network traffic, parsing data packets, monitoring network resource utilization, implementing network safety operating rules, identifying and analyzing network data, and diagnosing and repairing network problems, etc. It is the basis of a series of network functions. Because there are many different kinds of network protocol, and various new protocols emerge in endlessly, we need to parse a variety of new network protocols. Wireshark (formerly known as Ethereal) is a free open-source protocol parser, which is one of the most widely used online protocol parsing software in the world. Within the scope of the GNU GPL general license, users can obtain the software and the source code for free. As well, the researchers can customize or extend Wireshark according to its own needs.

The remainder of the paper is organized as follows. In Section 2, we describe the system structure of Wireshark. In Section 3, we present packet protocol analysis principle and development. In Section 4, we show an example of a plug-in protocol parser development and analyze the code. We end with concluding remarks in Section 5.

The System Structure of Wireshark

The system structure of Wireshark is shown in figure 1. The main function modules are as follows[1].

Figure 1. The system structure of Wireshark.
• GTK 1/2: A graphical window tool that controls all user input and output interfaces.
• Core: Connect other modules to play the role of comprehensive scheduling.
• Epan: The protocol analyzer of Wireshark
• Capture: The packet capture engine depends on the underlying Winpcap/libpcap library.
• Wiretap: The engine that reads and writes packet files from disk.

Packet Protocol Analysis Principle and Development

Principles of Protocol Parsing
Based on the OSI seven layer protocol model, the protocol data is sent from the top to the bottom, and the protocol parsing is the reverse, requiring bottom-up. Wireshark analyzes packet data in the form of a protocol tree. First of all, Wireshark makes a group package reduction after the protocol identification of the network layer, and then remove the network layer protocol header, giving the data inside to the transport layer for analysis. This has been going, to the application layer.

Figure 2 describes a HTTP packet parsing process in detail [2]. First, the data frame is obtained from Ethernet card, and the source/destination MAC address of the packet can be resolved. When the IP layer is reached, the IP protocol parsers are used to analyze the information of each field of IP header, and the payload data is presented to different sub-protocol parsers according to different packet types. In this case, the TCP protocol parser handles it. TCP resolves the payload content encapsulated in it to the HTTP parser module.

Development of Protocol Parsing
There are two ways to add a protocol parser to Wireshark: built-ins and plugins. The plug-in type is a plug-in (such as a Shared library /DLL) that registers itself in the main program for parsing [3]. The built-in type is to write a parser module that is compiled into the main program, which means it will always be available.

The difference between a plug-in and a built-in parser is small. However, the plug-in parser simply calls the interface based on the relevant instructions, without having to know the entire system's framework. But built-in type protocol parsing needs to be aware of the organizational structure of the Wireshark, which protocol parsing itself is developed in the form of built-in type, source files in what position, and how to compile and run, etc. So the built-in build cycle is much larger than the plug-in. Starting with a plug-in makes the initial development work easier, and the final code's deployment is similar to the built-in parser.

Code Analysis and Compilation of Protocol Parsers
An example of a plug-in protocol parser development is described below.
Code Analysis

A new protocol parser needs to be added to the main program through the registration protocol, the initial resolution parser, and the actual protocol parsing. Almost all protocol parsers in the Wireshark environment are done using these three steps [4-5].

In this paper, we will take a hypothetical "foo" protocol based on UDP as an example to implement the basic protocol parser step by step, demonstrating the whole addition process. This custom protocol contains the following basic items:

- packet type: 8 bytes, possible value: 1- initialization, 2- termination, 3- data.
- identification: 8 bytes, possible value: 0x01- start the packet, 0x02- end the packet, 0x04- priority packet.
- serial number: 32 bytes.
- IP address: 32 bytes.

The Registration Agreement. First, the proto_register_protocol() function is called to register the protocol, and its C language code is below.

```c
#include "config.h"
#include <epan/packet.h>
#define FOO_PORT 9877
static int proto_foo = -1;

void proto_register_foo(void)
{
    proto_foo = proto_register_protocol ("FOO Protocol", /* name */
                                             "FOO",      /* short name */
                                             "foo"       /* abbrev */);
}
```

Initial resolution analyzer. The protocol parsing function is associated with the main function by submitting the registered protocol to the main program.

```c
void proto_reg_handoff_foo(void)
{
    static dissector_handle_t foo_handle;
    foo_handle = create_dissector_handle(dissect_foo, proto_foo);
    dissector_add_uint("udp.port", FOO_PORT, foo_handle);
}
```

First, create a dissector handle, which is associated with the foo protocol and the function that performs the actual parsing. Next the handle is associated with the UDP port number so that the main program calls our parser when it sees UDP data on this port.

Actual protocol analysis. This function is used to parse the packets given to it, and the complexity of each protocol parser depends on the specific parsing of this section. For example, how to parse the header, how to parse the payload data, whether it is currently processed, or given to the corresponding child node to handle and so on.

```c
static void dissect_foo(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree)
{
    col_set_str(pinfo->cinfo, COL_PROTOCOL, "FOO");
    /* Clear out stuff in the info column */
    col_clear(pinfo->cinfo,COL_INFO);
}```
Packet data is placed in a special cache called TVB. The packet_info structure contains general data about the protocol, and we should update the information here. The tree parameter is where the details are resolved.

Compile

With the basic parser ready, you can compile and install it. In order to compile this parser and create a plug-in, you need to create some necessary supporting files in addition to the source code in packet-foo.c.

Take the foo protocol parser plug-in as an example to build this plug-in at least. The following files are established in the .../wireshark/plugins/foo directory:

- makefile.am: the Makefile template for UNIX/Linux.
- Makefile.common: contains the name of the plug-in file.
- makefile.nmake: a Makefile containing the Wireshark plug-in under the Windows platform.
- moduleinfo.h: contains the plug-in version information.
- moduleinfo.nmake: contains version information for the Windows platform dynamic link library (DLL).
- plugin.rc.in: contains DLL resource templates for Windows platforms.

When the supporting documents are ready and modified, using the terminal command CMD enters .../wireshark/plugins/foo directory, run nmake -f makefile.nmake XXX to compile, just like compiling wireshark source code. After compiling, generate the foo.dll, copy it to the compiled protocol plug-in directory, and perform the protocol parsing work.

The following is a simple udp sender and receiver to test the edited version of the foo protocol parsing plug-in. The parsing results are shown in figure 3, which verifies that the foo protocol parser is successfully registered and can be parsed accordingly. More in-depth and specific research and application can be done in the context of obtaining detailed information about protocol packets.

![Protocol analysis result diagram.](image)

Figure 3. Protocol analysis result diagram.

Conclusion

Nowadays, with the continuous popularization of network application and the updating of network technology, and new protocols are emerging endlessly, we require a good and extensible protocol analysis tool. Wireshark is highly regarded by more and more people for its good system architecture and open source. Developers do not need to understand the specific implementation of the system, and the existing framework of Wireshark can easily add a protocol parser of its own. It also provides powerful tools for network analysis work.
References


