Wireshark:
Automated generation of protocol dissectors
Project Requirements

by Erik Bergersen, Sondre Johan Mannsverk,
Terje Snarby, Even Wiik Thomassen, Lars Solvoll Tønder,
Sigurd Wien and Jaroslav Fibichr

September 7, 2011

Contents

Chapter 1

Use Cases

1.1 Actors

1.2 High-Level Use Case Diagram

1.3 Low-Level Use Case Diagrams

Chapter 2

Overview

2.1 Introduction

This document contains requirements for an utility that allows Wireshark to interpret the binary representations of C-language structs. While C structs seldom are exchanged across networks, they are sometimes used in inter-process communication. The purpose of the utility described here is to provide Wireshark with the capability of automatically dissecting the binary representation of a C struct, as long as its definition is known.

The expected work flow for the utility is to read one or more C header files, which contain struct definitions, and output Wireshark dissectors, implemented in Lua scripts. A configuration file or source code annotations in the header files may be used when additional configuration is required.

2.2 Requirements

An overview over both functional and non-functional requirements. See chapter 4 for more detailed description of the requirements.

FR01 Read C structs and generate Wireshark dissectors

FR02 Support structs with basic C data types

FR03 Support C preprocessor #include #define etc.

FR04 Recognize invalid values for struct members

FR05 A struct may have a header and/or trailer

FR06 Support also structs within structs

FR07 Support custom handling of specific data types

FR08 Support integers which indicate array of structs are following

FR09 Configuration for enumerated named value or a bit strings

FR10 Handle platform dependent endian and type sizes

NR01 Run on Windows 32bit & 64bit, Solaris 64bit and Sparc

NR02 Be able to run in batch mode

NR03 Need to have flexible configuration

NR04 The configuration needs to be well documented

Chapter 3

Prioritization

Chapter 4

Requirements

Table 4.1 lists the functional requirements and their priority, while Table 4.2 lists the non-functional requirements.

Table 4.1: Functional Requirements

ID	Description	Priority
FR01	The utility shall be able to read basic C language struct definitions, and generate a Wireshark dissector for the binary representation of the structs.	TODO
FR02	The utility shall support structs with the following basic data types: int, float, char, boolean, structs, unions, array and enums.	TODO
FR03	The utility must support the following C preprocessor directives and macros: `#include`, `#define`, `#if`, `WIN32`, `_WIN32`, `_WIN64`, `__sparc__`, `__sparc` and `sun`	TODO
FR04	The dissector shall be able to recognize invalid values for a struct member. Allowed ranges should be specified by configuration.	TODO
FR05	A struct may have a header and/or trailer (other registered protocol), which must be configurable.	TODO
FR06	The dissector shall be able to display each struct member. Structs within structs shall also be dissected and displayed.	TODO
FR07	Configuration must support custom handling of specific data types. E.g. a 'time_t' may be interpreted to contain a unixtime value, and be displayed as a date.	TODO
FR08	Configuration must support integer members which indicate that a variable number of other structs (array of structs) are following the current struct.	TODO
FR09	Configuration must support integer members which represent enumerated named value or a bit string.	TODO
FR10	The dissectors must be able to handle binary input which size and endian depends on originating platform.	TODO

Table 4.2: Non-Functional Requirements

ID	Description	Priority
NR01	The utility shall be able to run on Windows 32bit & 64bit, Solaris 64bit and Sparc.	TODO
NR02	The utility shall be able to run in batch mode.	TODO
NR03	The utility needs to have flexible configuration.	TODO
NR04	The configuration needs to be well documented.	TODO

Chapter 5

Test plan

TODO