Generating Wireshark Dissectors from XDR Files

Why you don't want to write them by hand

Richard Sharpe
Primary Data
Wireshark Core Team

8 November 2017
Agenda

- Motivation
- What We Built
- XDR Files (what they look like)
- How We Went About It
- Was It Successful?
- Next Steps
- Where Is The Code?
Motivation

- Writing dissectors is:
  - Tedious
  - Error prone
  - Requires lots of expertise
  - The last thing to be done in a project
- Engineers and QA demand them
- They change from time to time
What We Built

• **Two versions of the generator**
  - Second one in use now

• **Integrated with our build system**
  - Generates dissectors from all XDR files in the build
  - Builds wireshark with the extra dissectors
  - Packages it in RPMs
  - Every build (unfortunately increases build time lots)
XDR DR Files

- Describes a protocol
  - Constants
  - Enums
  - Data Structures
  - Typedefs
  - Functions/procedures
    - Arguments and return values
- \texttt{rpcgen} used to generate client and server stubs
• No executable statements in XDR

```c
#include "pd/types.h"
#include "pd/pd_dmc_mover_types.h"
#include "pd/nfsv3_xdr.h"

struct pddi_teardown_proxy_arg_t {
    pdx_job_id_t job_id;
    nfs_fh3 synth_fh;
};

const MAX_REQUEST = 10;

program PDDI_PROGRAM {
    version PDDI_RPC_V2 {
        /* NULL Procedure to test connectivity. */
        void PDDI_NULL(void) = 0;
        pddmc_job_res_t PDDI_DO_COPY(pddi_copy_arg_t a) = 3;
    } = 2;
} = 0x4D100000;
```
How We Went About It (HWWAI)

- Needed a parser for XDR
- Considered several approaches
  - Write one myself
  - Use Python
  - Other?
- Started with rpcgen from glibc
- Switched to the rpcgen code from tirpc
  - Essentially the original rpcgen
• With rpcgen's parser
  • No issues around compatibility!
  • Written in C
  • Could simply run through the Abstract Syntax Tree (if you can call it that.)

• Modified rpcgen a bit
  • Add dissector generator code (~2,000 LoC)

• Generate code for a dissector
• Not as simple as it seems
  • Writing code that generates code
  • The code generator has to compile
  • The generated code must compile
  • The resulting dissector must not crash
  • The resulting dissector must be correct
    • No undissected bytes
    • No incorrectly dissected bytes
• What experience did I have
  • Wrote a number of dissectors
  • Used a generator to create the original SMB dissector (Perl)
  • Lots of C experience
  • Willingness to push it to completion

• Had not much Wireshark for a long while
  • Lots had changed
• How long did it take
  • About 6 months part time for two versions
  • Including some time in Vancouver while on holidays
  • The rewrite was really needed
• How many dissectors do we generate?
  • 7-10 protocols
  • ~22,000 LoC in total
• Generate a new version of Wireshark
  • Stamped with build number and hash
• **Goal**
  
  • Generate code with no manual intervention

• **Overview of what we are generating**

<table>
<thead>
<tr>
<th>Boilerplate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declarations (hf, ett, etc)</td>
</tr>
<tr>
<td>Dissector code</td>
</tr>
<tr>
<td>Registration code hf array ett array etc</td>
</tr>
</tbody>
</table>
Look at a generated dissector

- Look at the generated code
A look at some results

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Source</th>
<th>XID</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2017-04-13</td>
<td>01:29:18.215220</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>270 127.0.0.1</td>
<td>0x99cb V1 PDDM_UPDATE_OBJ Call (Reply In 3)</td>
<td>12290 - 647 [ACK] Seq=1 Ack=205 Win=3080 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>2</td>
<td>2017-04-13</td>
<td>01:29:18.254816</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x99cb V1 PDDM_UPDATE_OBJ Call (Reply In 1)</td>
<td>12299 - 647 [ACK] Seq=205 Ack=29 Win=342 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>3</td>
<td>2017-04-13</td>
<td>01:29:18.254971</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>94 127.0.0.1</td>
<td>0x99cb V1 PDDM_UPDATE_OBJ Call (Reply In 1)</td>
<td>647 - 12299 [ACK] Seq=205 Ack=29 Win=342 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>4</td>
<td>2017-04-13</td>
<td>01:29:18.254984</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x99cb V1 PDDM_UPDATE_OBJ Call (Reply In 1)</td>
<td>647 - 12299 [ACK] Seq=205 Ack=29 Win=342 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>5</td>
<td>2017-04-13</td>
<td>01:29:26.032171</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>110 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 6)</td>
<td>0x8d8 V1 PDDM_PKG Call (Reply In 6)</td>
</tr>
<tr>
<td>6</td>
<td>2017-04-13</td>
<td>01:29:26.032371</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>94 127.0.0.1</td>
<td>0x97cb V1 PDDM_PKG Call (reply In 5)</td>
<td>0x8d8 V1 PDDM_PKG Call (Reply In 5)</td>
</tr>
<tr>
<td>7</td>
<td>2017-04-13</td>
<td>01:29:26.032392</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 5)</td>
<td>646 - 12299 [ACK] Seq=45 Ack=29 Win=342 Len=0 Tseq=33659118 TSval=0</td>
</tr>
<tr>
<td>8</td>
<td>2017-04-13</td>
<td>01:29:43.379845</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>270 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 10)</td>
<td>12299 - 647 [ACK] Seq=29 Ack=409 Win=3080 Len=0 Tseq=33712466 TSval=0</td>
</tr>
<tr>
<td>9</td>
<td>2017-04-13</td>
<td>01:29:43.379890</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 10)</td>
<td>646 - 12299 [ACK] Seq=45 Ack=29 Win=342 Len=0 Tseq=33659118 TSval=0</td>
</tr>
<tr>
<td>10</td>
<td>2017-04-13</td>
<td>01:29:43.381695</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>94 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 10)</td>
<td>12299 - 647 [ACK] Seq=29 Ack=409 Win=3080 Len=0 Tseq=33712466 TSval=0</td>
</tr>
<tr>
<td>11</td>
<td>2017-04-13</td>
<td>01:29:43.381707</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 10)</td>
<td>646 - 12299 [ACK] Seq=45 Ack=29 Win=342 Len=0 Tseq=33659118 TSval=0</td>
</tr>
<tr>
<td>12</td>
<td>2017-04-13</td>
<td>01:30:08.504405</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>270 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 13)</td>
<td>12290 - 647 [ACK] Seq=1 Ack=205 Win=3080 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>13</td>
<td>2017-04-13</td>
<td>01:30:08.502060</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>94 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 12)</td>
<td>12290 - 647 [ACK] Seq=1 Ack=205 Win=3080 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>14</td>
<td>2017-04-13</td>
<td>01:30:08.520689</td>
<td>127.0.0.1</td>
<td>TCP</td>
<td>66 127.0.0.1</td>
<td>0x97cb V1 PDDM_UPDATE_OBJ Call (Reply In 12)</td>
<td>12290 - 647 [ACK] Seq=1 Ack=205 Win=3080 Len=0 Tseq=33687341 TSval=0</td>
</tr>
<tr>
<td>15</td>
<td>2017-04-13</td>
<td>01:30:08.591455</td>
<td>127.0.0.1</td>
<td>PDDM_PROGR</td>
<td>110 127.0.0.1</td>
<td>0x97cb V1 PDDM_PKG Call (Reply In 14)</td>
<td>0x8d8 V1 PDDM_PKG Call (Reply In 14)</td>
</tr>
</tbody>
</table>
• Difficult parts of XDR
  • Include files
  • Individual declarations
    • Several different types
  • Unions
  • Recursive types (self relative)
Include files

```c
#include "pd/types.h"
#include "pd/pd_dmc_mover_types.h"
#include "pd/nfsv3_xdr.h"
```

- They started out as XDR files but are now .h files
- Converting name to .x file and searching for it
- Because we need the XDR file
- Must avoid generating code for definitions not used!
- Mark included definitions as reachable and unreachable
Review the data structures

- Look at the rpcgen data structures
  - What rpcgen uses to describe each XDR element
• Individual declarations

```c
union shr_client_match switch (some_type scm_type) {
    case SHR_CLIENT_MATCH_TYPE_IPV4:
        shr_ipgroup_range_ipv4 scm_range_ipv4;
    case SHR_CLIENT_MATCH_TYPE_IPV6:
        shr_ipgroup_range_ipv6 scm_range_ipv6;
    case SHR_CLIENT_MATCH_TYPE_HOSTNAME:
        shr_hostname scm_hostname;
    case SHR_CLIENT_MATCH_TYPE_NETGROUP:
        shr_netgroup scm_netgroup;
    default:
        void;
};
struct share_export7 {
    shr_security_flavors se_flavors<>;
    bool se_allow;
    shr_client_match se_clients<>;
    ...
};
```
First approach

- Several passes across the list of definitions from RPCGEN
- One for header field definitions
  - Used both for declarations and registration
- One for ETT definitions
- One for forward declarations
- One for dissecting structures
- One for the registration routine
- Etc
First approach, cont

- Used the linker to handle include files
  - Files without a program section were just a collection of dissection routines
- Became too hard to debug and keep correct
  - Because knowledge was distributed in many places
• Current approach

• Several passes across the list of definitions from RPCGEN

• Include file names converted to .x
  • Pulled in directly to the XDR token stream

• Pass across the definitions to mark reachable vs unreachable
  • Reachable from primary xdr file definitions
  • No code generated for unreachable definitions
Current approach, cont

- Build lists of structures
  - ETT variables
  - Header field definitions (everything needed)
  - Dissectors
  - Forward declarations
  - Etc
- Generate code from the lists in one pass
Code review

- Look at some generated code
- Look at parts of the generator
Integration into our build environment

Checks out the generator

Builds the generator

Very quick

Generates the dissectors

Very quick
• Writes their names to Custom.common
• Generates a hash of all the XDR files
• Modifies configure.ac and Makefile.am
  • Edits in extra version info from the hash
• Standard Wireshark build
  • Takes a long time
• Haven't bothered to use plugins
Look at the build script

• Some parts of the build
Was It Successful?

- Engineers scream if generation fails
- Engineers and QA depend on it
- Every build gets a new version of Wireshark
  - With the current dissectors
  - Could eliminate this step if no change in XDR files
- It just works
- So, yes, it has been successful!
Next Steps

- gRPC dissector generators
- Google's RPC language via protobufs
- Generators for other language-based protocol specifications
- Dissectors for Wi-Fi protocols etc
A dissector generator language

- For Wi-Fi dissectors?
- Use ANTLR4
  - Generate a parser from eBNF grammar
  - Add code generation in Java
    - ANTLR written in Java so easier
    - ANTLR makes writing grammars easy
  - Also makes generating code easy
typedef byte radio_id[6];

struct channel_preference {
    radio_id "Radio unique identifier";
    uint8 "Operating classes";
    channel_pref_detl "Operating class list"["Operating classes"];
};

protoDetails = { "IEEE 1905.1a", "ieee1905", "ieee1905" };

dissectorEntry ieee1905 = ieee1905_cmdu;

dissectorTable["ethertype"] = ieee1905;
Example ANTLR grammar

```antlr
grammar WiresharkGenerator;

protocol : protoDecl+ ;

protoDecl : dissectorTableDecl
| protoDetailsDecl
| dissectorEntryDecl
| enumDecl
| strenumDecl
| structDecl
| typedef
|

dissectorTableDecl : 'dissectorTable' '[' STRING ']' '=' ID ';' ;

protoDetailsDecl : 'protoDetails' '=' '{' STRING ',' STRING ',' STRING '}' ';' ;

...

structDecl : 'struct' ID '{' ( structEltDecl ';' )+ '}' ;

STRING : '"'.*?'"' ;  // Embedded quotes?
```
A look at the Java code

• Such as it is
What else can we do?

- **Generate Expert Info**
  - Recover from badly formatted fields
  - Flag incorrect values

- **Generate packet replay for testing**
  - scapy

- **Generate driver code as well?**
What else can we do, cont?

Specification → Generator

- Wireshark Dissector
- Packet Generator
- ?
Conclusions

• It can be a quick way to generate dissectors

• Correct code
  • As long as the generator is correct
  • My XDR generator took a while to get correct
    • Had to wait for engineers to use more features
Conclusions

- Want more features
  - Automatically add expert info
    - Malformed packets – point out malformed fields
    - Invalid values
  - All can be specified in the dissector spec
Where Is The Code?

- Gitlab
  
  https://gitlab.com/realrichardsharpe/wireshark_rpcgen.git
Questions?