SharkFest '17 Europe

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 Files

- Describes a protocol
  - Constants
  - Enums
  - Data Structures
  - Typedefs
  - Functions/procedures
    - Arguments and return values
- rpcgen used to generate client and server stubs

XDR files, cont

- No executable statements in XDR

```
%#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

HWWAI-2

- 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

HWWAI-3

- 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

HWWAI-4-1

- 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

HWWAI-4-2

- 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

HWWAI-6

- 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

HWWAI-5

- Goal
  - Generate code with no manual intervention
- Overview of what we are generating

Boilerplate
Declarations (hf, ett, etc)
Dissector code
Registration code hf array ett array etc

Look at a generated dissector

- Look at the generated code

A look at some results

pddm_packets_only.pcap [Wireshark 2.3.0-PD-8675309.cd4f68af (7efcae6 from mainline)]

File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help

Filter: Expression... Clear Apply Save

No.	Time	Destination	Protocol	Length	Source	XID	Info
1	2017-04-13 01:29:18.215220	127.0.0.1	PDDM_PROGR	270	127.0.0.1	0x99cb	V1 PDDM_UPDATE_OBJ Call (Reply In 3)
2	2017-04-13 01:29:18.254816	127.0.0.1	TCP	66	127.0.0.1		12299→647 [ACK] Seq=1 Ack=205 Win=3080 Len=0 TSval=33687341 TS
3	2017-04-13 01:29:18.254971	127.0.0.1	PDDM_PROGR	94	127.0.0.1	0x99cb	V1 PDDM_UPDATE_OBJ Reply (Call In 1)
4	2017-04-13 01:29:18.254984	127.0.0.1	TCP	66	127.0.0.1		647→12299 [ACK] Seq=205 Ack=29 Win=342 Len=0 TSval=33687341 TS
5	2017-04-13 01:29:26.032171	127.0.0.1	PDDM_PROGR	110	127.0.0.1	0xa8d9	V1 PDDM_PING Call (Reply In 6)
6	2017-04-13 01:29:26.032371	127.0.0.1	PDDM_PROGR	94	127.0.0.1	0xa8d9	V1 PDDM_PING Reply (Call In 5)
7	2017-04-13 01:29:26.032392	127.0.0.1	TCP	66	127.0.0.1		646→12299 [ACK] Seq=45 Ack=29 Win=342 Len=0 TSval=33695118 TSe
8	2017-04-13 01:29:43.379845	127.0.0.1	PDDM_PROGR	270	127.0.0.1	0x98cb	V1 PDDM_UPDATE_OBJ Call (Reply In 10)
9	2017-04-13 01:29:43.379890	127.0.0.1	TCP	66	127.0.0.1		12299→647 [ACK] Seq=29 Ack=409 Win=3080 Len=0 TSval=33712466 T
10	2017-04-13 01:29:43.381695	127.0.0.1	PDDM_PROGR	94	127.0.0.1	0x98cb	V1 PDDM_UPDATE_OBJ Reply (Call In 8)
11	2017-04-13 01:29:43.381707	127.0.0.1	TCP	66	127.0.0.1		647→12299 [ACK] Seq=409 Ack=57 Win=342 Len=0 TSval=33712467 TS
12	2017-04-13 01:30:08.504206	127.0.0.1	PDDM_PROGR	270	127.0.0.1	0x97cb	V1 PDDM_UPDATE_OBJ Call (Reply In 13)
13	2017-04-13 01:30:08.520660	127.0.0.1	PDDM_PROGR	94	127.0.0.1	0x97cb	V1 PDDM_UPDATE_OBJ Reply (Call In 12)
14	2017-04-13 01:30:08.520689	127.0.0.1	TCP	66	127.0.0.1		647→12299 [ACK] Seq=613 Ack=85 Win=342 Len=0 TSval=33737606 TS
15	2017-04-13 01:30:26.038145	127.0.0.1	PDDM_PROGR	110	127.0.0.1	0xa7d9	V1 PDDM_PING Call (Reply In 16)

```
▸ Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
▸ Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.0.1
▸ Transmission Control Protocol, Src Port: 647, Dst Port: 12299, Seq: 409, Ack: 57, Len: 204
▸ Remote Procedure Call, Type:Call XID:0x97cb380b
▾ PDDM_PROGRAM, PDDM_UPDATE_OBJ Call
    [Program Version: 1]
    [V1 Procedure: PDDM_UPDATE_OBJ (7)]
  ▸ trc_ctx
  ▸ share_event_id
  ▸ object_info
    inst_map_seq_id: 0
  ▸ online_instances<>
  ▸ archive_instances<>
  ▾ blob_data<>
      blob_data_len: 0
```

HWWAI-7

- Difficult parts of XDR
  - Include files
  - Individual declarations
    - Several different types
  - Unions
  - Recursive types (self relative)

HWWAI-8

- Include files

  %#include "pd/types.h"

  %#include "pd/pd_dmc_mover_types.h"

  - They started out as XDR files but are now .h files
  - Convenient to put them in search path

  %#include "pd/nfsv3_xdr.h"

  - 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

```
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<>;
...
```

HWWAI-10

- 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

HWWAI-11

- 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

HWWAI-12

- 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

HWWAI-13

- Current approach, cont
  - Build lists of structures
    - ETT variables
    - Header field definitions (every thing 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

HWWAI-14

- Integration into our build environment
  - Checks out the generator
  - Builds the generator
    - Very quick
  - Generates the dissectors
    - Very quick

HWWAI-15

- 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

Example dissector language

```
...

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

```
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?
```

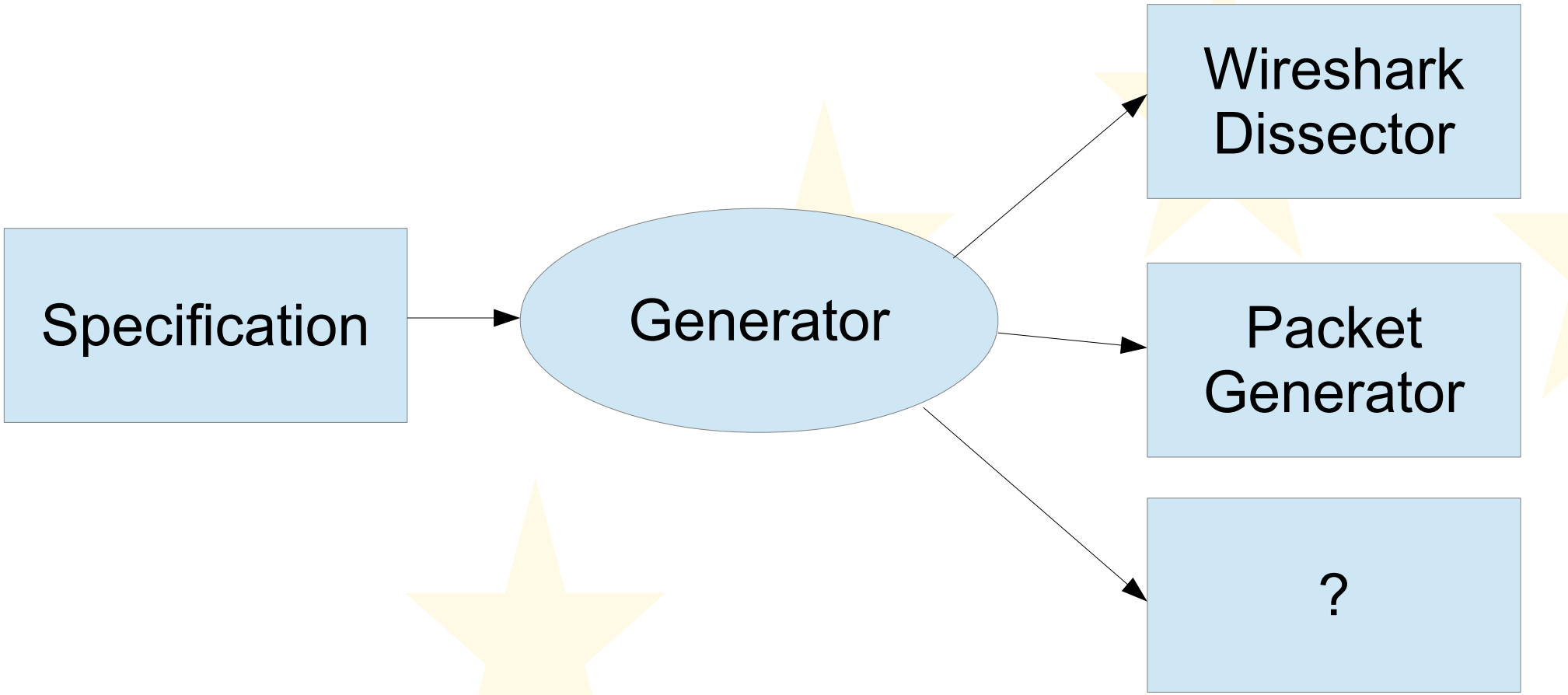
A look at the Java code

- 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?



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?