An Intrinsic Encoding of a Subset of C and its Application to TLS Network Packet Processing

Reynald Affeldt[1]
[1]National Institute of Advanced Industrial Science and Technology
e-mail : reynald.affeldt at aist.go.jp

TLS (a.k.a. SSL) is a widespread security protocol and errors in its implementation have disastrous consequences. For example, in 2014, the Heartbleed bug of OpenSSL allowed theft of private keys, thus compromising any security guarantee. To ensure the correctness of the implementation of TLS, programmers face two difficulties: an official specification with the ambiguities of natural language and error-prone low-level parsing of network packets. We report on the construction in the Coq proof-assistant [1] of libraries to formally model, specify, and verify C programs that process TLS packets. More precisely, we built an encoding of the core subset of C, a Separation logic that enables byte-level reasoning, and an encoding of a significant part of the official specification of TLS [2]. We also verified a parsing function of an existing implementation of TLS, namely PolarSSL [3]. We were able to correct bugs in the C source code and to spot ambiguities in the RFC.

1 ClientHello parsing: formalization

Our first library is an encoding of the core subset of C. Its originality is its use of dependent types (a feature of Coq) to provide an “intrinsic encoding”, i.e., an encoding such that only correctly-typed C programs can be represented.

Fig. 1 displays the first lines of the parsing function that we have formalized (`ssl_srv.c`, version 0.14.0). Variables and struct fields are strings (`"ret"`, `"in_hdr"`, etc.), prefixed with `%` when used as an expression or with `&→` when used to access fields of structs. Integer constants (`[ c ]`$_c$) are marked as unsigned or signed for clarity (`[ c ]`$_{uc}$ or `[ c ]`$_{sc}$). Variable assignment is marked as `:=`, or as `:=*` when the right-hand side is a pointer to be dereferenced. Logical operators appear as such (&, ≠). The intrinsic encoding guarantees at formalization-time that all the types agree.

```
Definition ssl_parse_client_hello :=
  "ret" := ssl_fetch_input(%"ssl",
                            [ 5 ]sc) ;
  If \b %"ret" ≠ [ 0 ]sc Then
    Return
  Else (
  "buf" :=* %"ssl" &→ "in_hdr" ;
  "buf0" :=* %"buf" ;
  If \b (%"buf0" & [ 128 ]uc) ≠
           [ 0 ]uc Then
    "ret" :=
      [ BAD_HS_CLIENT_HELLO ]c;
    Return
  Else ( ...
```

図 1. Formal model of the C function that parses TLS initialization packets (first lines only)

2 Formalization of the RFC of TLS

TLS enhances network applications by providing, on top of TCP, a cryptographic layer consisting of four protocols: packets from the Record protocol carry packets from the Handshake, Alert, or Change Cipher Spec protocols. The description of all these packet formats in the RFC is semi-formal: a dedicated syntax is introduced, but its use is not entirely consistent and many conditions remain described in prose.

Our work thus consisted in providing, as an alternative to the RFC, archetypal (but executable) Coq functions to recognize each kind of packets. For example, the payload of initialization packets are recognized by a function called `ClientHello` and the full initialization packets (with headers for the Record and Handshake layers) are recognized by a function called `RecordHandClientHello` (where `ClientHello` is used of course):

```
Definition RecordHandClientHello l :=
let (a1, l1) := TLSPlainText l in
let '(a2, m, l2) := Handshake l1 in
let (a3, l3) := ClientHello m l2 in
(a1 && a2 && a3, l3).
```

3 Specification with Separation logic

We also formalized in Coq a Separation logic for the core subset of C that we have introduced in Sect. 1. Unsurprisingly, specifications of C functions take the form of Hoare triples with a program and pre/post-conditions: `{ pre } prg { post }`

To write pre/post-conditions, we have formalized the connectives of Separation logic (most notably, the separating conjunction $\star$), which is handy when it comes to specify the mutable memory state of imperative programs (see [4] for details).

4 ClientHello parsing: specification

As for the case study, we verified the function from Sect. 1 by proving that, given any input from the network (modeled as a list of bytes `SI`), the function either fails (assertion `error` in Fig. 2) or succeeds (assertion `success`) in checking that the incoming ClientHello packet is valid. The most important point of the specification is the use of the formal specification `RecordHandClientHello` explained in Sect. 2 (see the postcondition of Fig. 2).

The precondition (lines 3–6) specifies the initial state: the initial value of the variable `"ssl"` (assertion `init_ssl_var`) and the initial state of the heap. The latter is captured by a Separation logic formula. The postcondition specifies also that the state of the heap after parsing has been updated correctly with the incoming data (lines 10–12). The last part of the specification is the assumptions (line 2) made by the programmer(s) of PolarSSL (the facts that it treats all minor version fields as “2” for “TLS version 1.1”, and it considers that the only compression method is “null” [2, §6.1]).

5 Results of the verification

We found several bugs in the course of verification that led us to patch the original C source code. In particular, checks performed by the `ssl_parse_client_hello` function were not sufficient: ill-formed packets might cause addressing of uninitialized memory and the absence of TLS extensions could not be checked.

It should be noted that above implementation errors and their fixes are similar to the Heartbleed bug found in OpenSSL (CVE-2014-0160) or the GnuTLS buggy parsing of session IDs (CVE-2014-3466). The programmers simply forgot to exhaustively check the size of payloads because their specifications in the RFC is not explicit enough.

This is our use of a precise model of the C language coupled with the use of Separation logic that ensures that we cannot miss the above errors when performing formal verification with Coq.

```
 0 Lemma POLAR_parse_client_hello_triple
 1  (SI : seq (int 8)) ...
 2  PolarSSLAssumptions SI →
 3  { init_ssl_var ⋆ init_bu ⋆
 4    init_rb ⋆ init_id ⋆
 5    init_ses ⋆ init_ciphers ⋆
 6    init_ssl_context }
 7  ssl_parse_client_hello
 8  { error ∨ (success ⋆
 9    !!((RecordHandClientHello SI).1)) ⋆
10    final_bu ⋆ final_rb ⋆
11    final_id ⋆ final_ses ⋆
12    init_ciphers ⋆
final_ssl_context }.
```

図 2. Formal specification of the C function that parses TLS initialization packets

謝辞 This work is partly supported by a Grant-in-Aid for Scientific Research (24500051).

参考文献

[1] The Coq Development Team. The Coq Proof Assistant: Reference Manual. INRIA, 2014.

[2] T. Dierks, E. Rescorla. The transport layer security (TLS) protocol version 1.2. Aug. 2008. IETF RFC 5246.

[3] PolarSSL. Available at `http://polarssl.org`

[4] R. Affeldt, K. Sakaguchi. An Intrinsic Encoding of a Subset of C and its Application to TLS Network Packet Processing. Journal of Formalized Reasoning 7(1):63–104, 2014