Securing Existing Software using Formally Verified Libraries

Tobias Reiher

FOSDEM, Brussels, 2020-02-01

Software Security
Security Vulnerabilities

- **CVE-1999-0015 – 5.0 MEDIUM – HP-UX, Windows, NetBSD, SunOS**
  - “Teardrop”
- **CVE-2014-0160 – 7.5 HIGH – OpenSSL**
  - “Heartbleed”: Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)
- **CVE-2017-0144 – 8.1 HIGH – Windows**
  - “EternalBlue”: Improper Input Validation (CWE-20)
- **CVE-2017-0785 – 6.5 MEDIUM – Android**
  - “BlueBorne”: Information Exposure (CWE-200)
- **CVE-2017-14315 – 7.5 HIGH – iOS**
  - “BlueBorne”: Improper Restriction of Operations within the Bounds of a Memory Buffer (CWE-119)
- **CVE-2018-10933 – 9.1 CRITICAL – libssh**
  - Improper Authentication (CWE-287)
- **CVE-2019-3560 – 7.5 HIGH – Fizz**
  - Loop with Unreachable Exit Condition (CWE-835)
- **CVE-2019-11477 – 7.5 HIGH – Linux**
  - Integer Overflow or Wraparound (CWE-190)

Integer Overflow in Fizz

- **Fizz** [1]
  - TLS 1.3 implementation by Facebook in C++
- **Vulnerability** [2]
  - Infinite loop triggered by unauthenticated remote attacker

1) https://github.com/facebookincubator/fizz 2) https://securitylab.github.com/research/facebook-fizz-CVE-2019-3560

Integer Overflow in Fizz

```
  while (true) {
...
        auto length = cursor.readBE<uint16_t>();
...         // assumption: length < 2**16 – 5, spec: length <= 2**14 + 256
        length +=
           sizeof(ContentType) + sizeof(ProtocolVersion) + sizeof(uint16_t);
        buf.trimStart(length);  // assumption: length > 0
        continue;
```


https://github.com/facebookincubator/fizz/commit/40bbb161e72fb609608d53b9d64c56bb961a6ee2

Software Security

Integer Overflow in Fizz

```
 static constexpr size_t kPlaintextHeaderSize =
     sizeof(ContentType) + sizeof(ProtocolVersion) + sizeof(uint16_t);
...
   while (true) {
...
         auto length = cursor.readBE<uint16_t>();
...
-        length +=
-           sizeof(ContentType) + sizeof(ProtocolVersion) + sizeof(uint16_t);
-        buf.trimStart(length);
+        buf.trimStart(static_cast<size_t>(kPlaintextHeaderSize) + length);
         continue;
```

How to prevent such bugs?

- **Software Quality Assurance** ⇨ Applied by Facebook
  - Code Reviews
  - Testing
  - Fuzzing
- **Static Code Analysis**
  - Variant Analysis ⇨ Applied by Semmle (acquired by GitHub) using CodeQL
  - Formal Verification

SPARK
Overview

- **Programming Language**
  - Based on Ada
  - Compilable with GCC and LLVM
  - Customizable runtime
  - Contracts (preconditions, postconditions, invariants)
- **Verification Toolset**
  - Absence of runtime errors
  - Functional correctness

- **Applications**
  - Avionics
  - Defense
  - Air Traffic Control
  - Space
  - Automotive
  - Medical Devices
  - Security

SPARK
Integer Overflow in Fizz

Componolit
Secure Systems Engineering

```
type UInt16 is range 0 .. 2**16 – 1;
```

```
...
        declare
           Length : UInt16 := Read_UInt16 (Cursor);
        begin
           Length := Length + 5;
           Trim_Start (Buf, Length);
...
```

SPARK
Integer Overflow in Fizz

```
type UInt16 is range 0 .. 2**16 – 1;
```

```
...
12          declare
13             Length : UInt16 := Read_UInt16 (Cursor);
14          begin
15             Length := Length + 5;
16             Trim_Start (Buf, Length);
...
```

```
Phase 1 of 2: generation of Global contracts ...
Phase 2 of 2: flow analysis and proof ...
plaintext_record_layer.adb:15:30: medium: range check might fail (e.g. when
Length = 65531)
```

Securing Existing Software

- **Current Situation**
  - Software usually written in unsafe languages (C, C++, …)
- **Migration to Language Supporting Formal Verification**
  - Very expensive when done manually
- **Options**
  - Only replace critical parts of software
  - Use code generation

RecordFlux
Toolset

- Formal Specification of Messages
- Model Verification
- Generation of Verifiable Binary Parsers
- Generation of Verifiable Message Generators

Securing Fizz using Verified Parser
Approach

- Creating TLS 1.3 specification (RFC 8446) using RecordFlux
- Generating parser based on specification
- Replacing Fizz’ parser
- Proving absence of runtime errors for SPARK implementation

Securing Fizz using Verified Parser

TLS 1.3 Message Specification

```
package TLS_Record is

  type Content_Type is (
    INVALID => 0,
    CHANGE_CIPHER_SPEC => 20,
    ALERT => 21,
    HANDSHAKE => 22,
    APPLICATION_DATA => 23,
    HEARTBEAT => 24
  ) with Size => 8;

  type Protocol_Version is (
    TLS_1_0 => 16#0301#,
    TLS_1_1 => 16#0302#,
    TLS_1_2 => 16#0303#,
    TLS_1_3 => 16#0304#
  ) with Size => 16;

  type Length is range 0 .. 2**14 + 256
    with Size => 16;
```

```
  type TLS_Record is message
    Tag : Content_Type;
    Record_Version : Protocol_Version;
    Length : Length
      then Fragment
        with Length => Length * 8
        if Tag /= APPLICATION_DATA
             and Length <= 2**14,
      then Encrypted_Record
        with Length => Length * 8
        if Tag /= APPLICATION_DATA
             and Record_Version = TLS_1_2;
    Fragment : Payload
      then null;
    Encrypted_Record : Payload;
  end message;

end TLS_Record;
```

Securing Fizz using Verified Parser

TLS 1.3 Message Specification

```
$ rflx check specs/tls_record.rflx

Parsing specs/tls_record.rflx... rflx: model
error: conflicting conditions for field
"Length" in "TLS_Record.TLS_Record" [...]
```

```
  type TLS_Record is message
    Tag : Content_Type;
    Record_Version : Protocol_Version;
    Length : Length
      then Fragment
        with Length => Length * 8
        if Tag /= APPLICATION_DATA
             and Length <= 2**14,
      then Encrypted_Record
        with Length => Length * 8
        if Tag /= APPLICATION_DATA
             and Record_Version = TLS_1_2;
    Fragment : Payload
      then null;
    Encrypted_Record : Payload;
  end message;

end TLS_Record;
```

Securing Fizz using Verified Parser
TLS 1.3 Message Specification

```
$ rflx check specs/tls_record.rflx

Parsing specs/tls_record.rflx... OK

$ rflx generate specs/tls_record.rflx

Parsing specs/tls_record.rflx... OK
Generating... OK
Created rflx-tls_record.ads
Created rflx-tls_record-
generic_tls_record.ads
Created rflx-tls_record-
generic_tls_record.adb
Created rflx-tls_record-tls_record.ads
Created rflx.ads
Created rflx-lemmas.ads
Created rflx-lemmas.adb
Created rflx-types.ads
Created rflx-types.adb
[...]
```

```
    type TLS_Record is message
      Tag : Content_Type;
      Record_Version : Protocol_Version;
      Length : Length
        then Fragment
          with Length => Length * 8
          if Tag /= APPLICATION_DATA
               and Length <= 2**14,
        then Encrypted_Record
          with Length => Length * 8
          if Tag = APPLICATION_DATA
               and Record_Version = TLS_1_2;
      Fragment : Payload
        then null;
      Encrypted_Record : Payload;
    end message;

end TLS_Record;
```

Securing Fizz using Verified Parser
Using Generated Parser

```
procedure Parse_Record_Message (Buffer :     RFLX.Types.Bytes;
                                Result : out CPP.Record_Record) is
   use TLS_Record.TLS_Record;
   Ctx : Context := Create;
begin
   Result := (Valid_Plaintext => CPP.Bool (False),
              Valid_Ciphertext => CPP.Bool (False),
              Tag => 0, Length => 0);
   Initialize (Ctx, Buffer);
   Verify_Message (Ctx);
   if Valid (Ctx, F_Length) then
      Result.Tag := CPP.Uint8_T (Convert (Get_Tag (Ctx)));
      Result.Length := CPP.Uint16_T (Get_Length (Ctx));
      if Valid (Ctx, F_Fragment) then
         Result.Valid_Plaintext := CPP.Bool (True);
      elsif Valid (Ctx, F_Encrypted_Record) then
         Result.Valid_Ciphertext := CPP.Bool (True);
      end if;
   end if;
end Parse_Record_Message;
```

* Adapted to API of RecordFlux 0.3 (original version used RecordFlux 0.1)

Securing Fizz using Verified Parser
SPARK to C Binding

```ada
type Record_Record is
   record
      Valid_Plaintext  : Bool;
      Valid_Ciphertext : Bool;
      Tag              : Uint8_T;
      Length           : Uint16_T;
   end record
  with
    Convention => C;

procedure Parse_Record_Message (
 Buffer_Address : System.Address;
 Buffer_Length  : Interfaces.C.Size_T;
 Result_Address : in out System.Address
) with
    Global => null,
    Export => True,
    Convention => C,
    External_Name =>
      "parseRecordMessage";
```

```c
struct RecordRecord {
    bool valid_plaintext;
    bool valid_ciphertext;
    uint8_t content_type;
    uint16_t length;
};

extern void parseRecordMessage(
    const uint8_t*,
    size_t,
    RecordRecord**
);
```

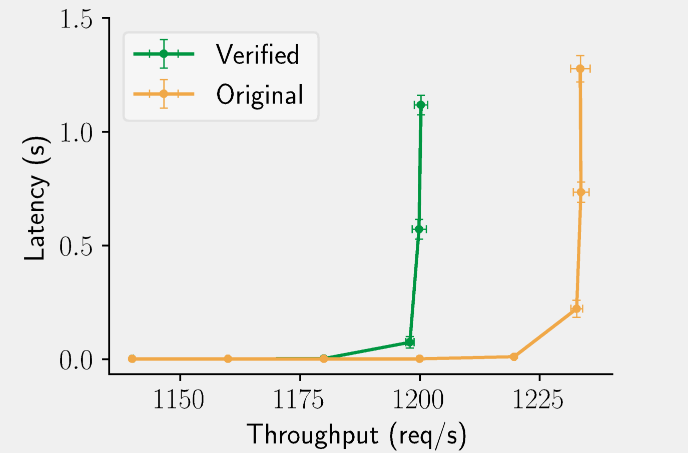
Securing Fizz using Verified Parser

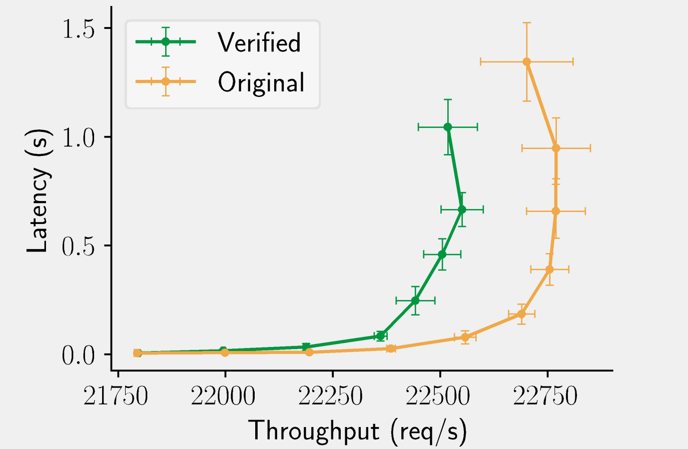
Integration into Fizz

```
+    Buf b;
+    cursor.cloneAtMost(b, 65536);
+    b->unshare();
+    if (b->isChained()) {
+      b->coalesce();
+    }
+
+    std::unique_ptr<RecordRecord> recordPtr(new RecordRecord());
+    RecordRecord *record = recordPtr.get();
+    parseRecordMessage(b->data(), b->length(), &record);
...
+      if (!record->valid_ciphertext && record->content_type == 0 && record->length == 0) {
+        throw FizzException("invalid encrypted record", AlertDescription::decode_error);
+      }
...
-        auto length = cursor.readBE<uint16_t>();
...
-        length +=
-            sizeof(ContentType) + sizeof(ProtocolVersion) + sizeof(uint16_t);
-        buf.trimStart(length);
+        buf.trimStart(kPlaintextHeaderSize + record->length);
```

Securing Fizz using Verified Parser
Evaluation

- **Testbed**
  - Sending continuously HTTP requests using modified wrk2
  - Handshake: TLS handshake for each request
  - Record: only one TLS handshake
- **Small performance impact**
  - Handshake: 2.7 % throughput loss (upper chart)
  - Record: 1.1 % throughput loss (lower chart)
  - Mostly due to conversions of data structures (SPARK records ⇨ C++ vectors and objects)





One Step Further
GreenTLS

- **Component-based high-assurance implementation of TLS 1.3**
- **Critical components in SPARK using RecordFlux**
- **Current State**
  - Complete message specification
  - Design and protocol specification in progress

https://github.com/Componolit/GreenTLS

Europäische Union
Europa fördert Sachsen.
ESF
Europäischer Sozialfonds
SACHSEN
Diese Maßnahme wird mitfinanziert durch Steuermittel auf Grundlage des von den Abgeordneten des Sächsischen Landtags beschlossenen Haushaltes.

Securing Existing Software
Conclusion

- Creation of formally verified library (parser for TLS 1.3)
- Integration of verified library into existing software (Fizz)
- Interaction between different programming languages feasible, but potentially cumbersome (C++ – SPARK)
- Low performance impact (1-3 %)

Questions?

Tobias Reiher
reiher@componolit.com

@Componolit · componolit.com · github.com/Componolit