Testing the security and reliability of Automotive Ethernet using fuzzing

Razvan Petre (razvan.petre@spirent.com)

Detroit, September 24th, 2019
Trends in Automotive Networks
Past, Present and Future of Automotive Architectures

Traditional or Central Gateway Automotive Architecture

Zonal-based Automotive Architecture

Domain-based Automotive Architecture

* text and diagrams taken from IEEE P802.1DG v0.3, July 2018
CAN vs Ethernet

Key differentiator: The Transport Network

• Standard Data Frame
  CAN 2.0A Frame Format

CAN High

<table>
<thead>
<tr>
<th>ECU 1</th>
<th>ECU 2</th>
<th>ECU 3</th>
<th>ECU n</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN Low</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

100Base-T1 TSN Switch

<table>
<thead>
<tr>
<th>ECU 1</th>
<th>ECU 2</th>
<th>ECU 3</th>
<th>ECU 4</th>
<th>ECU n</th>
</tr>
</thead>
<tbody>
<tr>
<td>CAN High</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

100Base-T1 TSN Switch

<table>
<thead>
<tr>
<th>Data (40 - 1500 bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet Type II Frame</td>
</tr>
<tr>
<td>(64 to 1518 bytes)</td>
</tr>
</tbody>
</table>

Ethernet Frame Format
TCP/IP and OSI Models

- Each layers communicates with its peer layer only
- Data encapsulation
Testing types, techniques and tactics

- Conformance testing
- Positive vs Negative testing
- Device vs System level testing
- Functional testing
- Performance: load, stress, robustness testing
- Security testing
- Black-box vs White-box testing

![Example Tests](e.g. OPEN TC8 v2.0, Aug 2017, 823 Tests)

- ARP (49 tests)
- ICMPv4 (22 tests)
- IPv4 (53 tests)
- Network Auto Configuration (56 tests)
- UDP (53 tests)
- TCP (222 tests)
- DHCPv4 (90 tests)
- SOME/IP Server (105 tests)
- SOME/IP ETS (144 tests)
What is fuzzing

• It is an **automated software testing technique**

• It involves providing **invalid, unexpected, or random data** as inputs to a computer program.

• An effective fuzzer generates semi-valid inputs that:
  – are "**valid enough**" to pass the parser, but
  – are "**invalid enough**" to expose corner cases
Who is using Fuzzing?

A brief history of fuzzing

• Testing programs with random inputs started in the **1950s**, when data was still stored on punched cards.
• In **1983**, "The Monkey“ was developed, a tool that would generate random inputs for classic Mac OS applications, such as MacPaint.
• The term "fuzzing" originates from a **1988** class project, taught by Barton Miller at the University of Wisconsin.
• In April **2012**, Google announced ClusterFuzz, a cloud-based fuzzing infrastructure for security-critical components of the Chromium web browser.
• In September **2016**, Microsoft announced Project Springfield, a cloud-based fuzz testing service for finding security critical bugs in software.
• In November **2018** Google revealed that OSS-Fuzz, the company's automated fuzzing service/bot, has identified and reported over 9,000 vulnerabilities in widely used open source projects since 2017.

IEEE-SA Ethernet & IP @ Automotive Technology Day, Sept 24th 2019, Detroit
# Types of Fuzzing

From dumb to smart fuzzers

<table>
<thead>
<tr>
<th>Dumb</th>
<th>Smart</th>
</tr>
</thead>
</table>

- **Random-based fuzzing**
  - No protocol knowledge

- **Data structure aware fuzzing**
  - A data model, a grammar, or a protocol description is required

- **Mutation-based fuzzing**
  - It generates inputs by modifying existing data (e.g. utilize a pcap capture as seed)

- **Generation-based fuzzing**
  - It uses fuzzing heuristics to generate data based on a specified model
Types of Fuzzing

...continued

• Black-box fuzzing
  – The fuzzer is unaware of internal program structure

• White-box fuzzing
  – The fuzzer leverages program analysis to systematically increase code coverage or to reach certain critical program locations

• Data fuzzing
  – Only the input data is modified, the order of the messages is always correct

• Behavioral fuzzing
  – Order or the messages is modified
  – Timing constraints can be tested as well
  – Illegal transitions in the protocol state machine
Fuzzing Automotive Ethernet Protocols

Choosing the right fuzzing ingredients for IVN testing

• Data aware fuzzing
• Both mutation- and generation- based fuzzing
• Black-box fuzzing
• Both data and behavioral fuzzing
Fuzzing Heuristics

The power of a smart fuzzer

Spirent Automotive Fuzzer offers:

Data Heuristics

- Strings: 30
- Numbers: 15
- Lists: 5
- Regular Expressions: 15

Behavior Heuristics

- Message Flow Order: 6
- Message Flow Timing: 4
- Protocol State Machine: 9
Spirent Automotive Fuzzer

A modularized solution for data and behavior fuzzing

1. Data Model
Contains the definition of the protocol messages
For each field, the fuzzing heuristic can be specified
Fields can be omitted from being fuzzed

2. Message Flow
Describes the message order and any timing restrictions between messages
Used for basic behavior fuzzing

3. Probes
How to check if the DUT is still alive

4. State machine
Used for advanced behavior fuzzing

5. Fuzzing Job
Uses all previous 4 components to glue together a fuzzing job
Data model

- Describes the structure of the protocol messages
- It is mandatory for the fuzzing engine
- Defines the rules for data fuzzing
  - “none”
  - “full”
  - “default”
  - “comma separated heuristic list”
Message flow

• A file describing:
  – order of the messages
  – any state information (relation between request / responses)
  – timing restrictions between messages

• It is optional and only required for behavior fuzzing
Probes

• Black-box testing requires a way to know if the DUT has crashed or not

• Sample probes available:
  – Ping
  – Create TCP connection
  – Terminate TCP connection
  – Receive [template] [after T1] [before T2]
  – Send [Eth Packet] [after T1] & Receive [template] [after T1] [before T2]
  – Run scripts
IPv4 fuzzing
Data fuzzing targeting the protocol stack

IPv4Header {
  version type="uint4"; fuzz="none",
  headerLength type="uint1"; fuzz="default",
  typeOfService {
    precedence type="uint3",
    delay type="bool",
    throughput type="bool",
    reliability type="bool",
    reserved type="uint2"; fuzz="none",
  }
  totalLength type="uint16"; fuzz="default",
  identification type="oct2"; fuzz="none",
  flags {
    reserved type="bool"; fuzz="full",
    doNotFragmentFlag type="bool"; fuzz="full",
    moreFlagsFlag type="bool"; fuzz="full",
  },
  fragmentOffset type="uint13"; fuzz="default",
  ...
}
DoIP fuzzing

Behavior fuzzing targeting the protocol stack & application

Message Flow

Tester Application

Vehicle announcement
TCP Sync
TCP Sync + Ack
TCP Ack
Routing Activation Request
Routing Activation Response

max 50ms
max 2ms
max 50ms

DoIP GW
ARXML – AUTOSAR XML

Description of communication networks

- A file used to represent the configuration of AUTOSAR based ECUs
- It contains description for
  - Clusters
  - ECUs
  - Frames
  - PDUs
  - Signals
  - Data Types
- Similar to DBC, LDF or FIBEX files
ARXML

Fuzzing directly from ARXML

Spirent ARXML Parser

Data Model

Message Flow

Probes

Fuzzing Job

Spirent Fuzzing Engine

Fuzzed Data

DUT ECU

Automated Operation

Optional Operation
Fuzzing CAN based applications

AUTOSAR Time Sync over CAN
Fuzzing CAN

```
AutosarSpec {  
    type type=oct1; value=0x20; fuzz="none",  
    crc type=oct1; value=function_crc; fuzz="none",  
    timeDomain type=uint4; fuzz="full",  
    seqCounter type=uint4; fuzz="default",  
    userDefined type=oct1; value=0x0; fuzz="none",  
    secondPart type=uint32; fuzz="default"  
}
```

AUTOSAR Specification of Time Synchronization over CAN

Summary
Best practices & benefits of using fuzzing for automotive industry

• Fuzzing saves time & money, as minimal user input is required
• Fuzzing does not replace the need for conformance, performance or security testing; it rather complements them and brings much better coverage
• Same test methodology and tools can be used for fuzzing Ethernet protocols from L2 to L7, as well as legacy automotive busses
• Data Fuzzing can be used for any protocol
• Behavior Fuzzing for protocols with complex state machines is not recommended, since there is a considerable user input required
• Good Fuzzing Heuristics must be used to reduce the number of generated messages, since ECUs usually have limited resources