Architecture and Protocol Verification and Attack Analysis

Ludovic Apvrille

Institut Telecom, Telecom ParisTech
ludovic.apvrille@telecom-paristech.fr

July, 1, 2010
Outline

Introduction
- Context
- Performance and attack analysis
- Our Toolkit: TTool

Performance analysis
- DIPLODOCUS
- Case study: Active Brake

Attack analysis
- TURTLE
- Case study: Needham-Schoreder

Outlook
Outline

Introduction
  Context
  Performance and attack analysis
  Our Toolkit: TTtool

Performance analysis

Attack analysis

Outlook
On-board Vehicle Systems

On-board vehicle system

- ECUs (Electronic Control Units) = set of hardware components
  - Execution elements (CPUs, HWAs)
  - Communication elements (e.g., busses)
  - Storage elements (e.g., RAM, flash)
  - I/O devices, including sensors / actuators

- Software components
  - Executed on CPUs

One of EVITA’s goals:
Proving security properties on those systems
Proving Security Properties: Overall Methodology

**Methodology**

1. Requirement identification
2. Architecture specification
3. Specification of security-related protocols
4. Verification of security properties on the overall system (Architecture + protocols)
   - Performance analysis
   - Attack analysis

**Objective of this demonstration**

- Focus on the last stage (verification)
Proving Security Properties: Overall Methodology (Cont’d)

Performance evaluation

- Impact of security mechanisms on system performance

Attack analysis

- **Magnified view approach**
  - Proof of security properties on a subpart of the EVITA architecture (e.g., a given protocol).

- **Global composition approach**
  - Reuse of proofs performed on sub-elements to validate requirements over the full system
  - Next presentation
Issues

(1) Performance properties

- Impact of the EVITA security architecture on system performance?
  - Cryptographic algorithms and protocols
- Partitioning issue
  - Shall algorithms be software or hardware implemented? Distributed among ECUs or centralized in a given ECU

(2) Security properties

- Security requirements have been previously identified
- Derive attacks from requirements and ...
- Prove that those attacks are not possible in the EVITA infrastructure
Modeling and Verification Approach

Objective

- Performance evaluation, Attack analysis (magnified view approach)

- Consider inputs (e.g., EVITA deliverables)
- Make a model, using e.g. SysML and UML models
- Verify properties using simulation or formal verification techniques
**Modeling and Verification Approach (Cont’d)**

<table>
<thead>
<tr>
<th>Analysis</th>
<th>(1) Performance analysis</th>
<th>(2) Attack analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Profile</td>
<td>DIPLODOCUS</td>
<td>TURTLE</td>
</tr>
<tr>
<td>Verification technique</td>
<td>Simulation</td>
<td>Formal verification (model-checking)</td>
</tr>
<tr>
<td>Focus of the model</td>
<td>Application complexity and architecture elements</td>
<td>Protocol description and basic architecture elements. Attacks modeling</td>
</tr>
<tr>
<td>Tools</td>
<td>TTool (edition, simulator)</td>
<td>TTool, CADP, UPPAAL</td>
</tr>
</tbody>
</table>
TTool: Main Features

- Open-source UML toolkit
- Meant to support UML2 profiles
  - 8 UML profiles are currently supported
    - e.g., TURTLE, DIPLODOCUS
- Mostly programmed in Java
  - Editor, interfaces with external tools
  - Simulators are programmed in C++ or SystemC
- Formal verification and simulation features
  - Hides formal verification and simulation complexity to modelers
  - Relies on external tools
  - Press-button approach
TTool: TURTLE and DIPLODOCUS

TTool: TURTLE and DIPLODOCUS

Analysis of formal verification and simulation results
Graph analysis
Diagram animation
Waveform analysis

Introduction
Performance analysis
Attack analysis
Outlook
Context
Performance and attack analysis
Our Toolkit: TTool

Ludovic Apvrille
Architecture / Protocols / Attacks
Outline

Introduction

Performance analysis
  DIPLODOCUS
  Case study: Active Brake

Attack analysis

Outlook
DIPLODOCUS in a Nutshell

DIPLODOCUS = UML Profile

- System-level Design Space Exploration
- Y-Methodology
- MARTE compliant

Main features

- Data are abstracted
- Formal semantics
- Very fast simulation support
- Fully supported by an open-source toolkit
  - TTool
DIPLODOCUS: Methodology for Performance Evaluation (Cont’d)
Description of the Active Brake Use Case

- Message sent from one car to another car (car2car)
  - Immediate danger of collision
  - Instant brake manoeuvre
- Message received and checked at Communication Unit level
- Plausibility check at Chassis Safety Controller level
  - If braking is the best solution, a brake order is sent to the brake control unit
    - Power Train Controller is also informed (to decelerate, etc.).
  - Braking information might be forwarded to other neighbour cars
Application Modeling
Architecture Modeling and Mapping
<table>
<thead>
<tr>
<th>CPU</th>
<th>Load</th>
<th>Contention delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load_Emulation</td>
<td>0.15711</td>
<td>29973</td>
</tr>
<tr>
<td>CPU_CU</td>
<td>0.11244</td>
<td>0</td>
</tr>
<tr>
<td>HSM_CU</td>
<td>0.11939</td>
<td>0</td>
</tr>
<tr>
<td>CPU_BCU</td>
<td>0.00010</td>
<td>6806</td>
</tr>
<tr>
<td>HSM_BCU</td>
<td>0.00004</td>
<td>0</td>
</tr>
<tr>
<td>CPU_PTC</td>
<td>0.00018</td>
<td>0</td>
</tr>
<tr>
<td>CPU_ChassisSensor</td>
<td>0.00035</td>
<td>200000</td>
</tr>
<tr>
<td>CPU_EnvSensor</td>
<td>0.01115</td>
<td>5818</td>
</tr>
<tr>
<td>HSM_CSC</td>
<td>0.11827</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### A Few Simulation Results (Cont’d)

#### Buses

<table>
<thead>
<tr>
<th>Bus</th>
<th>Load</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCU_local.Bus</td>
<td>0.00017</td>
</tr>
<tr>
<td>CSC_local.Bus</td>
<td>0.56926</td>
</tr>
<tr>
<td>PTC_local.Bus</td>
<td>0.00026</td>
</tr>
<tr>
<td>CU_local.Bus</td>
<td>0.55783</td>
</tr>
<tr>
<td>CU_SOC_Bus</td>
<td>0.78811</td>
</tr>
<tr>
<td>Main_CAN</td>
<td>0.71469</td>
</tr>
<tr>
<td>CSC_SOC_bus</td>
<td>0.74216</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Outline

Introduction

Performance analysis

Attack analysis

TURTLE
Case study: Needham-Schoreder

Outlook
TURTLE = UML Profile

- Targets temporally constrained embedded systems
- Three sub-profiles: analysis, design, deployment
- Formal verification (and simulation)
- TURTLE Design = class diagram + a set activity diagrams

Main features

- Non deterministic operators
  - Choice, delays
- Fully supported by an open-source toolkit
  - TTool
TURTLE: Methodology for Attack Analysis

- Attack trees (T2200)
- Use Cases (T2100)
- Architecture specification (T3200)
- Specification of cryptographic protocols (T3300)
- Prolog-based toolkit of Eurecom (T3300)

TURTLE: Methodology for Attack Analysis

- Requirements (T3300) [SysML Requirement Diagrams]
- System Design (T3400) [TURTLE class and activity diagrams]
- Formal Proof (T3400) [Based on model-checking techniques, And external toolkits (CADP, UPPAAL)]
- System analysis (T3400) [TURTLE Sequence Diagrams]

Ludovic Apvrille
Model: Main Principles

Modeled elements

- Hardware elements in ECUs
  - HSM
  - Communication networks
- Software elements
  - Protocol stack at involved ECUs

Proving security properties

- Observer technique
- Model-checking is used to search for a given action
Description of the Case Study

Why this case study (not directly related to EVITA)?

- Illustrate proofs of security requirements with TURTLE
- A small yet representative system
- Contains all interesting concepts:
  - Entities, network elements, crypto functions and protocols, attacks

Description

- Alice and Bob, who want to exchange a confidential data
- Use the Needham-Schroeder protocol to setup a session key K, using a trusted server
- Then, Bob sends the data to Alice using K
The Needham-Schroeder Protocol

**Description**

A represents Alice, B Bob, S the Server; \( R_X \) is a random number generated by \( X \), and \( K_{XY} \) a key used by \( X \) and \( Y \) to cipher / decipher information with a symmetric encryption algorithm.

1. \( A \rightarrow S : A, B, R_A \)
2. \( S \rightarrow A : \{ R_A, B, K_{AB}, \{ K_{AB}, A \} _{K_{BS}} \} _{K_{AS}} \)
3. \( A \rightarrow B : \{ K_{AB}, A \} _{K_{BS}} \)
4. \( B \rightarrow A : \{ R_B \} _{K_{AB}} \)
5. \( A \rightarrow B : \{ R_B - 1 \} _{K_{AB}} \)

**Requirement req1**

The data sent by Bob to Alice shall be confidential.
Attacks on the Needham-Schroeder Protocol

- Several known attacks against Needham-Schroeder

(Cx denotes an attacker pretending to be an entity x):

1. \( A \rightarrow C_S : A, B, R_A \)
2. \( C_B \rightarrow S : B, A, R_C \)
3. \( S \rightarrow C_B : \{ R_C, A, K_{BA}, \{ K_{BA}, B \} K_{AS} \} K_{BS} \)
4. \( C_A \rightarrow B : \{ R_C, A, K_{BA}, \{ K_{BA}, B \} K_{AS} \} K_{BS} \)
5. \( B \rightarrow C_A : \{ R_B \} R_C \)
6. \( C_A \rightarrow B : \{ R_B - 1 \} R_C \)

- From that attack, req1 can be proved as non-satisfied.
Class Diagram
Activity Diagram of Alice

- pdu.origination, pdu.destination=0; pdu.data0=16; pdu.data1=16; pdu.data2=ra
- socket_outpdu
- socket_in?pdu
- cipherpdu/KAB?pdu

- [not(pdu.data0 == ra) and (pdu.data1 == 16)]
  - serverCanBeTrusted
  - KAB = pdu.data2
  - socket_outpdu
  - socket_in?pdu
  - cipherpdu/KAB?pdu

- pdu.data0 = pdu.data3-1
  - cipherpdu/KAB?pdu
  - pdu.destination=pdu.origination; pdu.origination

- socket_outpdu
  - socket_in?pdu
  - cipherpdu/KAB?pdu

- [not(pdu.data0 == privateDataOfBob)]
  - dataOut

- [not(pdu.data0 == privateDataOfBob)]
  - dataOut

- Deciphering data: if this is equal to private data of Bob, then the data transferred.

- Forging "A, B, Ra"
  - Trying to decipher with key KAB

- Forging "(Kab, AjKAB) from previous PDU"

- Waiting for a challenge

- Deciphering server's challenge, then decrement number, re-encrypt with KAB, and send back "(Ra-1)KAB" to sender

- Waiting for data
Activity Diagram of Attacker
Formal Verification with CADP

Verification approach
- Generate a Reachability Graph using CADP
- Minimize of the reachability graph
- Search for traces containing the \textit{attackOK} and \textit{attackKO} actions

Reachability graph

Bottom: the initial state
Top: the final state

- \textit{cypher<2,3,103,20,369,569,221>}
- \textit{cypher<2,3,103,20,369,569,221,19,1>}
- \textit{cypher<2,3,84,1,350,550,202>}
- \textit{attackOK}
Formal Verification with UPPAAL

Verification approach
- Select actions of interest on the UML model
- Automatically invoke UPPAAL
- Search the accessibility and liveness of selected actions

Network can be probed

- Reachability of: Action state (attackKO) -> property is NOT satisfied
- Reachability of: Action state (attackOK) -> property is satisfied
- Reachability of: Action state (dataKO) -> property is NOT satisfied
- Reachability of: Action state (dataOK) -> property is satisfied
- Liveness of: Action state (attackKO) -> property is NOT satisfied
- Liveness of: Action state (attackOK) -> property is NOT satisfied
- Liveness of: Action state (dataKO) -> property is NOT satisfied
- Liveness of: Action state (dataOK) -> property is NOT satisfied
## Formal Verification with UPPAAL (Cont.)

<table>
<thead>
<tr>
<th>Network cannot be probed</th>
<th>Network is always probed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reachability of:</strong> Action state (attackKO)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Reachability of:</strong> Action state (attackOK)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Reachability of:</strong> Action state (dataKO)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Reachability of:</strong> Action state (dataOK)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Liveness of:</strong> Action state (attackKO)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Liveness of:</strong> Action state (attackOK)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Liveness of:</strong> Action state (dataKO)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is NOT satisfied</td>
<td></td>
</tr>
<tr>
<td><strong>Liveness of:</strong> Action state (dataOK)</td>
<td></td>
</tr>
<tr>
<td>-&gt; property is satisfied</td>
<td></td>
</tr>
</tbody>
</table>
Outline

Introduction

Performance analysis

Attack analysis

Outlook
Results

Fully integrated environment for the design and verification of embedded systems

- Based on UML / SysML, open-source toolkit (TTool)
- Formal proof can address
  - Safety and security properties
    - Proofs achieved on authenticity, confidentiality, freshness
  - Functional and non functional properties

Recall on methodological stages

- Requirement capture (SysML, DIPLODOCUS)
  - Attack trees, definition and organization of requirements
- Performance analysis (DIPLODOCUS)
- Attack analysis, magnified view approach (TURTLE)
A Few Industrial Case Studies with TTool

- MPEG coders and decoders (Texas Instruments)
- LTE (Freescale)
- Partitioning in vehicle embedded systems, formal proof of security properties (EVITA project)
- Many other systems!
To Go Further ...

**TTool**
- Type *TTool UML* under google
- And click on the *I am lucky* button!

**DIPLODOCUS, TURTLE**
- *DIPLODOCUS UML*
- *TURTLE UML*