AVATAR: A SysML Environment for the Formal Verification of Safety and Security Properties

G. Pedroza    L. Apvrille    D. Knorreck

System-on-Chip laboratory (LabSoC), Institut Telecom
Telecom ParisTech, LTCI CNRS

9-12 May / NOTERE 2011
1. Introduction
   - Motivations

2. AVATAR SysML environment
   - AVATAR Profile
   - Security extensions

3. Security Proofs
   - ProVerif
   - Transformation rules
   - Formal verification

4. Case Study
   - Keying Protocol modeling
   - Verification and results
Tendency in embedded systems design

**Distributed Systems**
- Networked
- Public channels
- Security critical

Require **security** properties

**Local Systems**
- Encapsulated
- Protected channels
- Safety critical

Require **safety** properties

**Distributed systems**
- Security critical

**Local systems**
- Safety critical

Require **safety and security** properties

Formal Verification
Modeling approaches target either:
- safety properties or
- security properties

Safety and security models should be maintained

Consistency problems
Methodology Phases

Overview

Requirements capture: Models requirements to be satisfied, e.g. with Requirement Diagrams

System analysis: Analysis of system behavior, e.g. using Sequence Diagrams

System design: Captures system behavior, e.g. with AVATAR profile

Property modeling: Captures properties to be verified, e.g. in TEPE Diagrams

Formal verification: Formal proves over targeted properties

Refinement: Repeat previous stages adding system elements up to final design

1 See [1]
Methodology Phases

Implementation

Requirement capture
(Requirement Diagrams)

Attack trees
(Parametric Diagrams)

Use cases and interfaces
(Use Case Diagrams)

Scenarios
(Activity Diagrams, Sequence Diagrams)

Requirement

Analysis

Simulation
Formal verification

Design (Block Definition Diagram, Internal Block Diagram)

Detailed design
(State Machine Diagrams)

Property modeling
(Extended Parametric Diagrams - TEPE)

Design

Simulation
Formal verification
Code generation

Gabriel Pedroza
AVATAR for safety and security formal proofs
AVATAR
In a Nutshell

- SysML environment supporting all methodological phases
- Graphical capture of properties
- Integrated simulation
- Safety and **security** proofs at the push of a button
- C-POSIX code generation
- SysML Block Definition and Internal Block Diagrams
- Block = attributes, methods, in/out signals, behaviour
Block’s behaviour is described in terms of SysML State Machine Diagrams

- Non deterministic choices
- Non deterministic temporal operators
Property Modeling

Security properties

- Customized Parametric Diagrams (TEPE)

![Diagram](image-url)
Profile limitations to address security

**Initial knowledge:** No way to preshare data between blocks

**Cryptographic functions:** Not predefined and should be modelled

**Communication Architecture:** Channels can not be eavesdropped

**Attacker model:** Not included and not easily representable

**Security properties:** Not easily representable
Initial knowledge: Introduced as a common knowledge by the pragma:

```
#InitialCommonKnowledge Alice.sk Bob.sk
```

Cryptographic functions: Predefined in each crypto block: \(MAC()\), \(encrypt()\), \(decrypt()\), \(sign()\), \(verifyMAC()\), \(verifySign()\)...

Communication Architecture: Common broadcast channels can be defined in blocks. Attackers can eavesdrop public ones

Attacker model: Taken from the underlying security framework \(ProVerif^2\)
Security properties: **Confidentiality** of data intended to be secret is captured in the pragma:

\#Confidentiality Alice.sk

**Authenticity** of a block exchange is captured in the pragma:

\#Authenticity Alice.sendingMessage.m1

Bob.messageDecrypted.m2

\[
\begin{align*}
\text{makingMessage} & \quad m\cdot\text{data} = \text{secretData} \\
\text{sendingMessage} & \quad m1 = \text{encrypt}(m, sk) \\
\text{chout}(m1) & \quad \text{a} - \text{Alice} \\
\text{waitingForMessage} & \quad \text{b} - \text{Bob} \\
\text{messageDecrypt} & \quad m = \text{decrypt}(m2, sk) \\
\text{messageDecrypted} & \quad \text{receivedData} = m\cdot\text{data}
\end{align*}
\]
Why ProVerif?
as underlying formal framework

ProVerif is \(^3\) ...

- quite generic: targets communicating systems modeling in general
- completely automated
- well suited for communicating entities (CEs) modeling:
  - based upon process algebras
  - CEs represented as pi-processes
- oriented to prove security properties:
  - confidentiality
  - authenticity
- endowed with an attacker targeting security properties
- supported by a rigorous formal approach
- implemented with a resolution algorithm

\(^3\) See [2]
### Syntax of the process calculus

<table>
<thead>
<tr>
<th>M, N ::=</th>
<th>terms</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y, z</td>
<td>variables</td>
</tr>
<tr>
<td>a, b, c, k</td>
<td>names</td>
</tr>
<tr>
<td>f(M₁...M₇)</td>
<td>constructor application</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>P, Q ::=</th>
<th>processes</th>
</tr>
</thead>
<tbody>
<tr>
<td>out(c,M); P</td>
<td>outputs M in c then P</td>
</tr>
<tr>
<td>in(c,M); P</td>
<td>inputs M from c then P</td>
</tr>
<tr>
<td>new a; P</td>
<td>defines a restricted to P</td>
</tr>
<tr>
<td>event myEvt(x); P</td>
<td>executes an event myEvt(x) then P</td>
</tr>
<tr>
<td>let x=g(M₁...M₇) in P</td>
<td>destructor application</td>
</tr>
<tr>
<td>else Q</td>
<td>conditional</td>
</tr>
<tr>
<td>if M=N then P else Q</td>
<td>parallel composition of processes P, Q</td>
</tr>
<tr>
<td>P</td>
<td>Q</td>
</tr>
<tr>
<td>!P</td>
<td>null process</td>
</tr>
<tr>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

---

4 See [3]
Defined as queries:\footnote{See [3]}

**Confidentiality:** can the attacker disclose secret data - *mySecret*?

query attacker:*mySecret*

**Authenticity:** can the attacker break the receiver-sender correspondence?

query evinj:eventReceiveM(x)\(\Rightarrow\)evinj:eventSendM(x)
### AVATAR block diagram to ProVerif

#### Translation rules

<table>
<thead>
<tr>
<th>AVATAR</th>
<th>ProVerif</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block declaration</td>
<td><code>let myBlock_0 =</code></td>
<td>The initial process <code>myBlock_0</code></td>
</tr>
<tr>
<td>Block data types</td>
<td><code>new data1;</code></td>
<td>The new name <code>data1</code></td>
</tr>
<tr>
<td>Block input signal</td>
<td><code>free ch1.</code></td>
<td>The input channel <code>ch1</code>. Quite similar rule for <code>output</code> or <code>private</code> channel</td>
</tr>
<tr>
<td>Common knowledge</td>
<td><code>new val;</code></td>
<td>The static variable <code>val</code> is only known by the processes <code>Block_i</code>, <code>i=1...n</code></td>
</tr>
<tr>
<td>Confidentiality pragma</td>
<td><code>query attacker: data.</code></td>
<td><code>data</code> confidentiality will be proved</td>
</tr>
</tbody>
</table>
Performed at a push of a button!
Here the process⁶:

1. Translation of AVATAR model to ProVerif
2. Gained attacker knowledge in form of Horn clauses
3. **Confidentiality:** for each query attacker:mySecret the attacker:
   - builds a finite inference space of horn clauses
   - searches whether mySecret can be inferred
4. **Authenticity:** for each query
   evinj:eventReceiveM(x)==>evinj:eventSendM(x) the attacker:
   - test all input channels
   - acts on behalf of sender - or receiver - in replicated sessions
   - builds a finite space of horn clauses
   - proves sender-receiver correspondence

⁶See [4]
Security Goal: distribute a new secret key amongst members of a group of in-car Electronic Control Units (ECUs).\(^7\)

\(^7\) See [5], [6]
Each ECU associated to a crypto block
- Public input and output channels
- Predefined crypto functions are used: `sencrypt()`, `sdecrypt`, `MAC()`, etc.
- Confidentiality of SesK and mutual authentication `<auth>` of ECUs is required.
Keying Protocol verification results\(^8\)

<table>
<thead>
<tr>
<th>Verification Scheme</th>
<th>AVATAR model and pragmas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeled Blocks</td>
<td>ECU1, ECUKM, ECUN</td>
</tr>
<tr>
<td>Verified confidential data</td>
<td>SesK, PSK-1, PSK-N</td>
</tr>
<tr>
<td>Verified authenticity correspondences</td>
<td>ECU1&lt;auth&gt;ECUKM, ECUKM&lt;auth&gt;ECUN</td>
</tr>
<tr>
<td>RESULT #Confidentiality B.dat</td>
<td>True for each verified data (dat)</td>
</tr>
<tr>
<td>RESULT #Authenticity B(_d).Send.Mx B(_o).Valid.Mx</td>
<td>True for each correspondence and message (Mx)</td>
</tr>
<tr>
<td>Observations</td>
<td>Keying Protocol preserves data confidentiality and authenticity.</td>
</tr>
</tbody>
</table>

\(^8\)See [7]
Conclusions

AVATAR:

- eases embedded systems modeling
- easily proves safety and security properties
- avoids models consistency maintainability
- is fully supported by TTool\(^9\)
- has been tested in industrial projects

However:

- only targets confidentiality and authenticity
- not suitable for temporal analyses
- richer notions of attackers are required

\(^9\)See [8]
Next steps to do

- introduce temporal analyses capabilities
- introduce richer notions of attackers:
  - to prove message integrity
  - to prove message freshness
- automatic code generation from models
- code maintainability
- adapt tool support
Thanks


“The EVITA european project.”
http://www.evita-project.org/.

