Secure Information Flow in Programs

Prof. Cinzia Bernardeschi
Department of Information Engineering
University of Pisa, Italy
cinzia.bernardeschi@unipi.it

May 7-10, 2019 – Thessaloniki, Greece

Outline

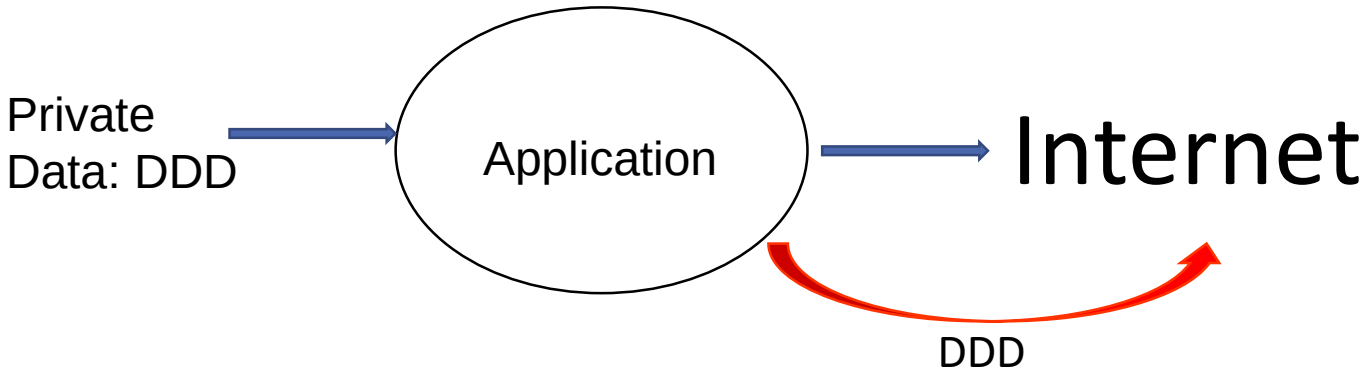
- Background
  - Data leakage
  - Multi-level Security policy
  - Information flow in programs
  - Examples of illegal flow of information
- A static analysis approach for secure information flow checking
- A case study: secure flow in AUTOSAR models
- Conclusions

Data leakage

- private data made publicly available
- interference between private and public data (information on private data revealed indirectly)
- colluding applications for data leakage

Information flow analysis

Private data made publicly available



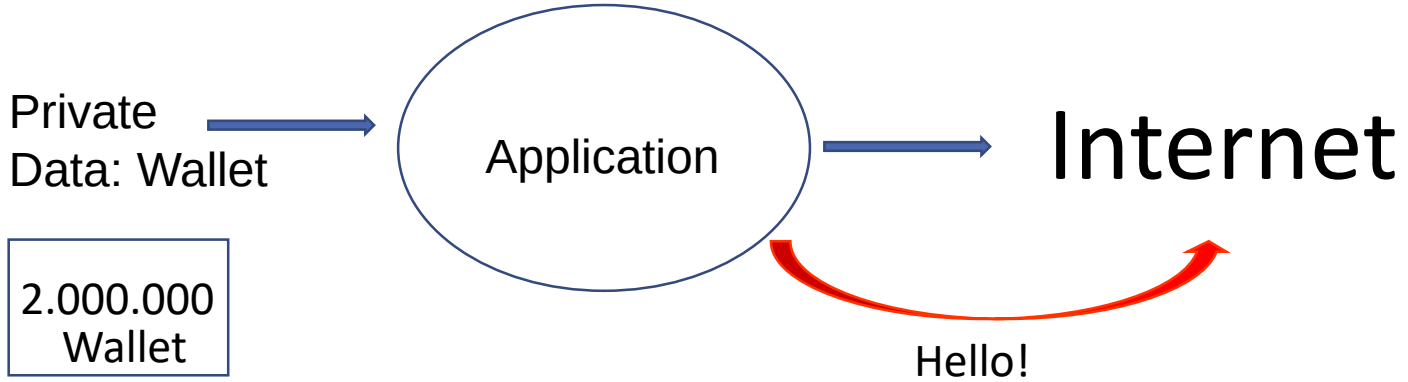
Secure information flow is violated

Application authorized to access private data
Application authorized to access Internet

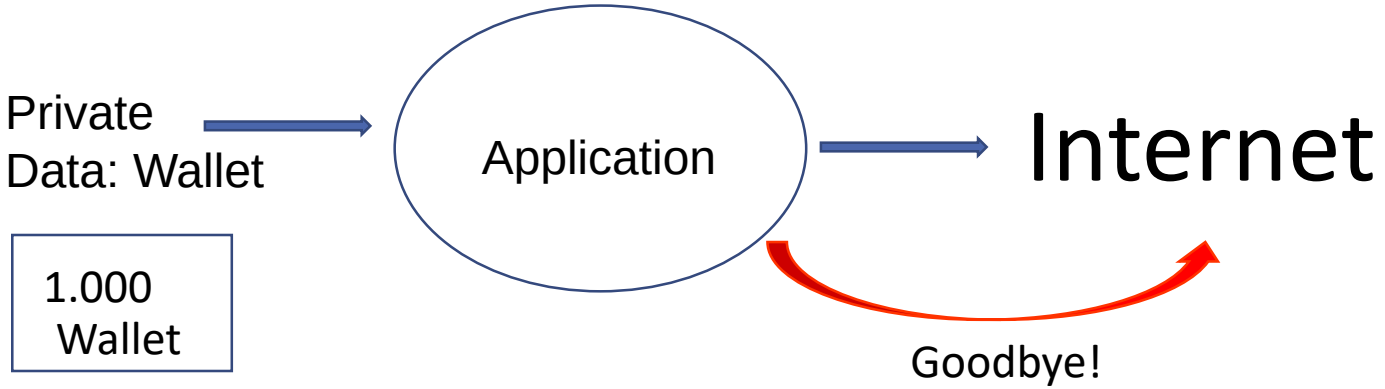
Control on the information sent on Internet!!!!!

Limit of Firewall and Access control mechanisms

Interference between private and public data

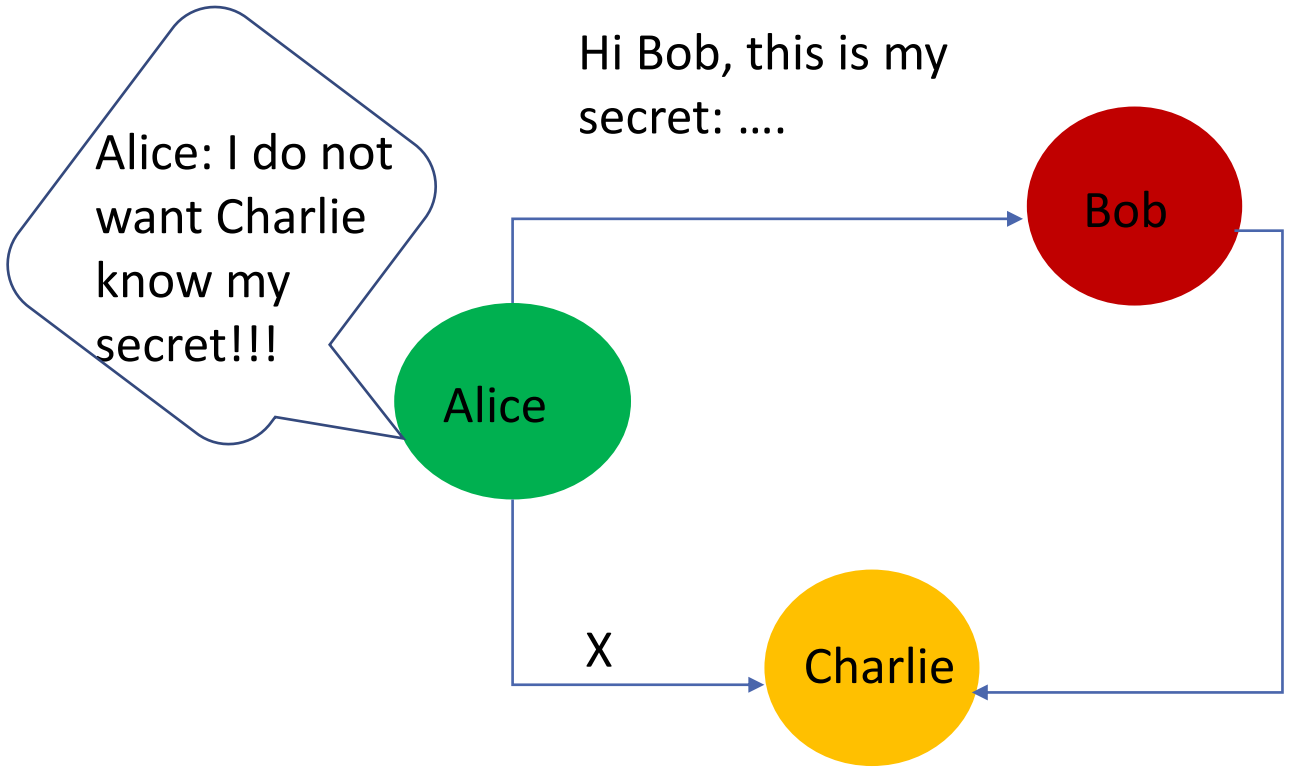


Secure information flow is violated



if Wallet > 1.000.000 then
«Hello!»
else «Goodbye!»

Colluding applications for data leakage



Secure information flow is violated

Colluding applets

April 2017

The team reports that the types of app fall into two major categories / *Justin Sullivan/Getty Images*

The biggest security risks can come from some of the least capable apps

***The Independent* (British online newspaper)**

Taken from: http://www.independent.co.uk/life-style/gadgets-and-tech/news/android-app-steal-users-data-colluding-each-other-research-cartel-information-a7663976.html

Multi-level Security policy

UNIVERSITÀ DI PISA

- a security policy that allows the classification of **data** and **users** based on a system of hierarchical security levels

private

public

- Inputs and outputs are classified as either public (*low* sensitive) or private (*high* sensitive). A program has the **non-interference** property if and only if any sequence of low inputs will produce the same low outputs, regardless of the high level inputs.

- the program responds in exactly the same manner on low outputs whether or not high sensitive data are changed. The low user will not be able to acquire any information about data and the activities (if any) of the high user.

Multi-level Security policy

Non-interference property

the security domain private is non-interfering with domain public if no input by private can influence subsequent outputs seen by public.

Secure information flow property

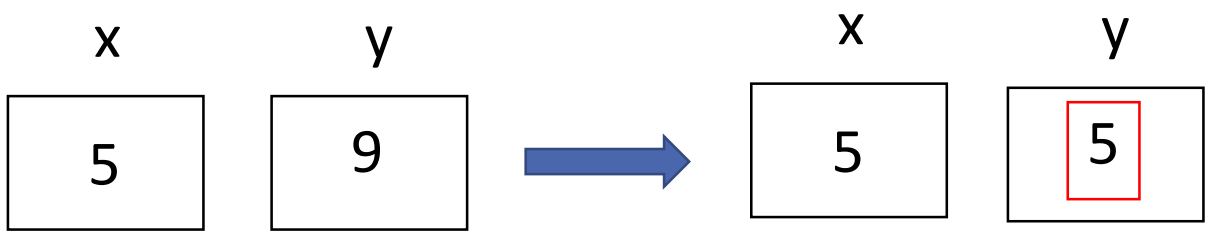
Basics of information flow

Simple high-level language

Let x, y be variables

y := x; explicit flow

variable y is assigned the value of x; there is an explicit flow x to y



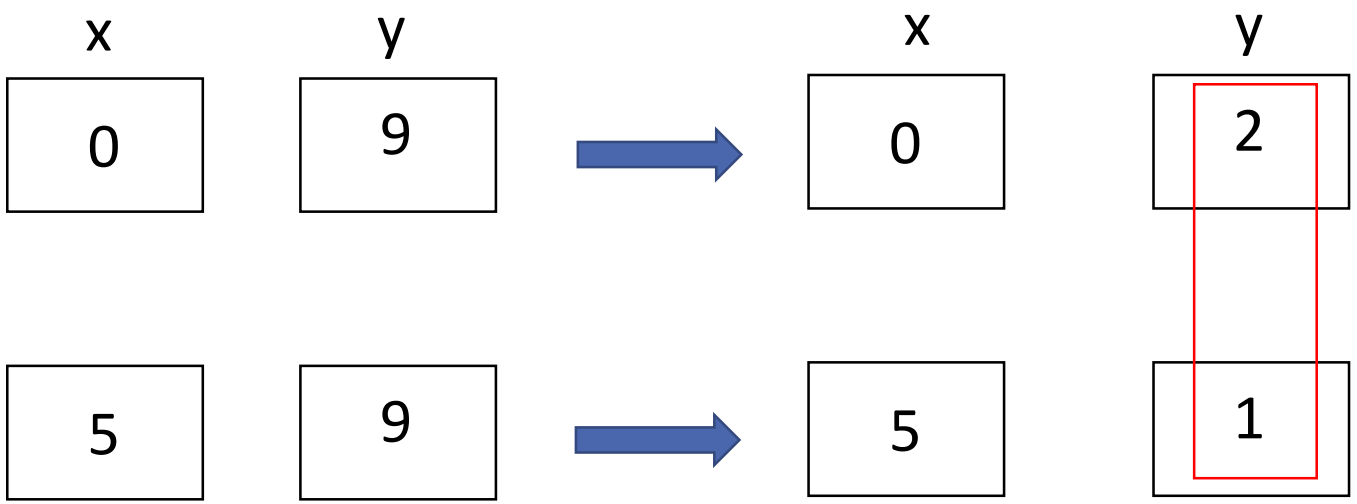
The final value of y reveals the value of x

Basics of information flow

```
if  (x = 0)
        then y := 2;
         else y := 1;
```

implicit flow

implicit flow from variable x to y, since y is assigned different values depending on the value of the condition of the control instruction (variable x)



Observing the final value of y reveals information on the value of x

Information flow: implicit flow

A conditional instruction in a program causes the beginning of an implicit flow.

The implicit flow begins when the conditional instruction starts (we have an opened implicit flow);

All the instructions in the scope of the if depend on the condition of the if.

Secure Information Flow

- a program P
- a lattice of security levels $\mathcal{L}$
- every variable of P is assigned a security level in $\mathcal{L}$
- A program P satisfies Secure Information Flow if information at a given security level does not flow to lower levels

Lattice

Let be given a set A and order relation $\leq$ on A.

$(A, \leq)$ is a lattice if every pair of elements in A has both a greatest lower bound (*glb*) and a least upper bound (*lub*).

D. E. Denning, P. J. Denning. Certification of programs for secure information flow. Communications of the ACM, 20(7), 1977

Secure Information Flow

$\mathcal{L}$ = {L, H }, with L ≤ H L: public, H: private

Let x: H, y: L

H

|

L

- Explicit information flow

  y := x;

- Implicit information flow

  if (x = 0) then y:=2; else y:=1

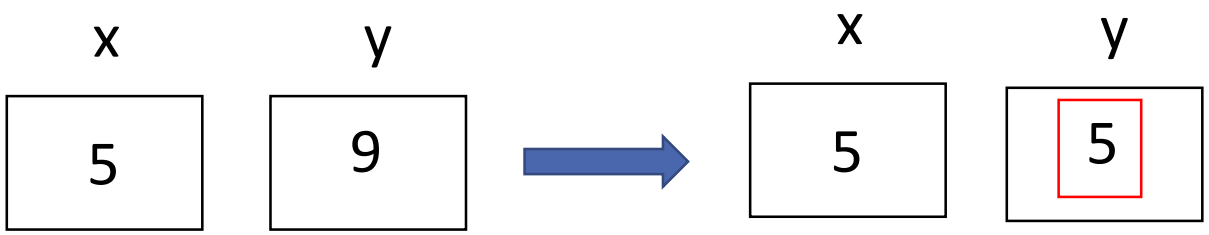
SIF: the final value of each variable does not depend on the initial value of variables with higher level

Secure Information Flow violation

Let x: H, y: L

y := x;

explicit flow



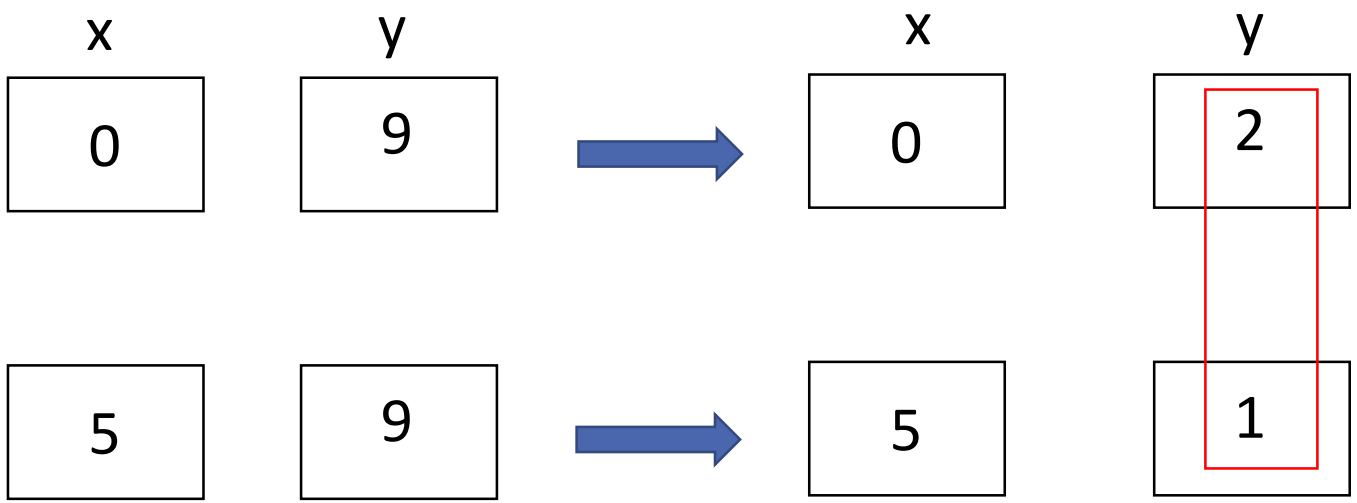
Anyone can see the value of the high sensitive variable x !!!

Secure Information Flow violation

Let x: H, y: L

```
if (x = 0) then y := 2;
           else y := 1;
```

implicit flow



Anyone can infer information on the value of the high sensitive variable x !!!

Secure Information Flow checking

Typing approach: the security information of a variable belongs to its type, and secure Information flow is checked by means of a type system. Hierarchy between types. Types = H, L

Semantic-based approach: execute the program

Abstract interpretation approach: execute the program on abstract domains

An advantage of 3) with respect to those based on 1) is that it is semantics based and thus keeps information on the dynamic behavior of programs, allowing to check more precisely the desired properties.

```
y=x;
y=0;
```

rejected by 1)

```
if 0 then y=x; else skip;
```

rejected by 1) and by 3)

Abstract Interpretation of the Operational semantics

UNIVERSITÀ DI PISA

- Definition of a concrete instrumented semantics recording the information flow (collecting semantics)
- Definition of an abstract semantics taking only what concerns the information flow
- Proof of correctness of the abstraction

Basics of Operational semantics

$$exp ::= const \mid var \mid exp\ op\ exp$$
$$com ::= var := exp \mid \texttt{if}\ exp\ \texttt{then}\ com\ \texttt{else}\ com \mid \texttt{while}\ exp\ \texttt{do}\ com \mid com\ ;\ com \mid \texttt{skip}$$

m = [(x, 1) (y, 0)]

P
c1:y:=7;
c2: if (x=0) y:=2; else y:=5;

constants V = {k, k', ….}
memory m: var → V
c = com1; com2; …; comj

m: [(x, k) (y, k') ….]
state: <c, m>

Q^{ε} = set of states
$\rightarrow^{\varepsilon} \subseteq Q^{\varepsilon} \times Q^{\varepsilon}$ transition system

Transition system

<c1;c2, [(x, 1) (y, 0)]>
↓ ε
<c2 , [(x, 1) (y,7)]>
↓ ε
< - , [(x, 1) (y, 5) ….]>

Operational semantics

$$\textbf{Expr}_{const}\ \frac{}{\langle k, m\rangle \longrightarrow^{\epsilon}_{expr} k} \qquad \textbf{Expr}_{var}\ \frac{}{\langle x, m\rangle \longrightarrow^{\epsilon}_{expr} m(x)}$$

$$\textbf{Expr}_{op}\ \frac{\langle e_1, m\rangle \longrightarrow^{\epsilon}_{expr} k_1 \quad \langle e_2, m\rangle \longrightarrow^{\epsilon}_{expr} k_2 \quad k_1\ op\ k_2 = k_3}{\langle (e_1\ op\ e_2), m\rangle \longrightarrow^{\epsilon}_{expr} k_3}$$

$$\textbf{Ass}\ \frac{\langle e, m\rangle \longrightarrow^{\epsilon}_{expr} k}{\langle x{:=}e, m\rangle \longrightarrow^{\epsilon} m[k/x]} \qquad \textbf{Skip}\ \frac{}{\langle \texttt{skip}, m\rangle \longrightarrow^{\epsilon} \langle m\rangle}$$

$$\textbf{If}_{true}\ \frac{\langle e, m\rangle \longrightarrow^{\epsilon}_{expr} true}{\langle \texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2, m\rangle \longrightarrow^{\epsilon} \langle c_1, m\rangle}$$

$$\textbf{While}_{true}\ \frac{\langle e, m\rangle \longrightarrow^{\epsilon}_{expr} true}{\langle \texttt{while}\ e\ \texttt{do}\ c, m\rangle \longrightarrow^{\epsilon} \langle c; \texttt{while}\ e\ \texttt{do}\ c, m\rangle}$$

$$\textbf{While}_{false}\ \frac{\langle e, m\rangle \longrightarrow^{\epsilon}_{expr} false}{\langle \texttt{while}\ e\ \texttt{do}\ c, m\rangle \longrightarrow^{\epsilon} \langle m\rangle}$$

$$\textbf{Seq}_1\ \frac{\langle c_1, m\rangle \longrightarrow^{\epsilon} \langle m'\rangle}{\langle c_1;c_2, m\rangle \longrightarrow^{\epsilon} \langle c_2, m'\rangle} \qquad \textbf{Seq}_2\ \frac{\langle c_1, m\rangle \longrightarrow^{\epsilon} \langle c_2, m'\rangle}{\langle c_1;c_3, m\rangle \longrightarrow^{\epsilon} \langle c_2;c_3, m'\rangle}$$

Our approach

- We attach a security level σ to each data k.
- During the execution of a program, σ indicates the least upper bound of the security levels of the information flows, both explicit and implicit, on which k depends.
- To deal with implicit flow, the concept of execution environment is introduced

Concrete Operational sematics

An instrumented semantics which:

- Handles values (k, σ) annotated with a security level.
  During the execution of a program, σ indicates the least upper bound of the security levels of the information flows, both explicit and implicit, on which k depends.

- Executes instructions under a security environment σ. During the execution, σ represents the least upper bound of the security levels of the open implicit flows. σ is (possibly) upgraded when a branching instruction begins and is (possibly) downgraded when all branches join.

C(P) : concrete transition system for a program P

Concrete Operational sematics

Security levels	$\mathcal{L}$ = {L < H}	$\sigma, \tau, ..$
Constants	V	k, k', ..
Concrete Values	$\mathcal{V}$ = V × $\mathcal{L}$	(k, σ)
Concrete Memories	**M** = var → $\mathcal{V}$	M, M',..
Environments	$\mathcal{L}$	$\sigma, \tau, ..$

M : [(x, (k, σ)) (y, (k', τ)) ....]

c = com_1; com_2; ...; com_j

state: <c^{σ}, M>

where

σ is the execution environment

Q = set of states

→ $\subseteq$ QxQ transition system

Concrete Operational sematics

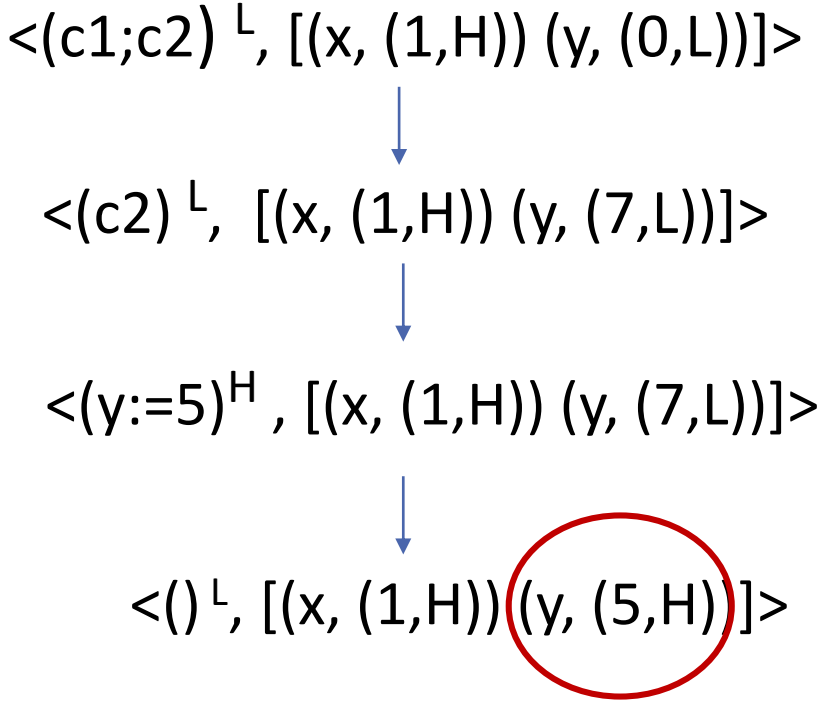
Let x:H, y:L

m = [(x, (1,H) (y, (0,L))]

initial execution environment: L

P
c1: y:=7;
c2: if (x=0) y:=2; else y:=5;

Concrete transition system C(P)



Secure information flow violation

Concrete Operational sematics

Let x:H, y:L

m = [(x, (1,H) (y, (0,L))]

Initial execution environment: L

P
c1: x:=7;
c2: if (x=0) y:=2; else y:=5;

Concrete transition system C(P)

<(c1;c2) ^{L}, [(x, (1,H)) (y, (0,L))]>

↓

<(c2) ^{L}, [(x, (7,L)) (y, (0,L))]>

↓

<(y:=5)^{L} , [(x, (7,L)) (y, (0,L))]>

↓

<() ^{L}, [(x, (7,L)) (y, (5,L))]>

Concrete Operational sematics

$$\textbf{Expr}_{const}\ \frac{}{\langle k^{\sigma}, M\rangle \longrightarrow_{expr} (k, \sigma)} \qquad \textbf{Expr}_{var}\ \frac{M(x) = (k, \tau)}{\langle x^{\sigma}, M\rangle \longrightarrow_{expr} (k, \sigma \sqcup \tau)}$$

$$\textbf{Expr}_{op}\ \frac{\langle e_1^{\sigma}, M\rangle \longrightarrow_{expr} (k_1, \tau_1)\ \ \langle e_2^{\sigma}, M\rangle \longrightarrow_{expr} (k_2, \tau_2)}{\langle (e_1\ op\ e_2)^{\sigma}, M\rangle \longrightarrow_{expr} (k_1\ \ op\ k_2, \tau_1 \sqcup \tau_2)}$$

$$\textbf{Ass}\ \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} v}{\langle (x{:=}e)^{\sigma}, M\rangle \longrightarrow M[v/x]} \qquad \textbf{Skip}\ \frac{}{\langle \texttt{skip}^{\sigma}, M\rangle \longrightarrow \langle M\rangle}$$

$$\textbf{If}_{true}\ \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (true, \tau)}{\langle (\texttt{if}\ e\ \texttt{then}\ c_1\ \texttt{else}\ c_2)^{\sigma}, M\rangle \longrightarrow \langle c_1^{\tau}, Impl(M, Mod(c_1) \cup Mod(c_2), \tau)\rangle}$$

$$\textbf{While}_{true}\ \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (true, \tau)}{\langle (\texttt{while}\ e\ \texttt{do}\ c)^{\sigma}, M\rangle \longrightarrow \langle (c;\texttt{while}\ e\ \texttt{do}\ c)^{\tau}, Impl(M, Mod(c), \tau)\rangle}$$

$$\textbf{While}_{false}\ \frac{\langle e^{\sigma}, M\rangle \longrightarrow_{expr} (false, \tau)}{\langle (\texttt{while}\ e\ \texttt{do}\ c)^{\tau}, M\rangle \longrightarrow \langle Impl(M, Mod(c), \tau)\rangle}$$

$$\textbf{Seq}_1\ \frac{\langle c_1^{\sigma}, M\rangle \longrightarrow \langle M'\rangle}{\langle c_1^{\sigma};w, M\rangle \longrightarrow \langle w, M'\rangle} \qquad \textbf{Seq}_2\ \frac{\langle c_1^{\sigma}, M\rangle \longrightarrow \langle w', M'\rangle}{\langle c_1^{\sigma};w, M\rangle \longrightarrow \langle w';w, M'\rangle}$$

Abstract Operational semantics

- abstracts concrete values into their security level: $\alpha(k, \sigma) = \sigma$
- uses the same rules of the concrete semantics on the abstract domains

A(P) : abstract transition system for program P
- finite
- multiple path
- each path of C(P) is correctly abstracted onto a path of A(P)

A program P has secure information flow if in each final state of A(P), each x: σ holds a value $\tau \leq \sigma$

Abstract Operational sematics

Abstract security levels	$\mathcal{L}^{\#} = \mathcal{L}$	$\sigma, \tau, ..$
Abstract constants	$V^{\#}$	{ . }
Abstract Values	$\mathcal{V}^{\#} = \mathcal{L}$	(σ)
Abstract Memories	$\mathbf{M}^{\#} = \text{var} \rightarrow \mathcal{V}^{\#}$	$M^{\#}, M^{\#'},..$
Environments	$\mathcal{L}^{\#}$	$\sigma, \tau, ..$

$M^{\#}$: [(x, σ) (y, τ) …]

c = com_1; com_2; …; com_j

state: <c^{σ}, $M^{\#}$ >

where

σ is the execution environment

$Q^{\#}$ = set of states

$\rightarrow \subseteq Q^{\#} \times Q^{\#}$ transition system

Abstract Operational sematics

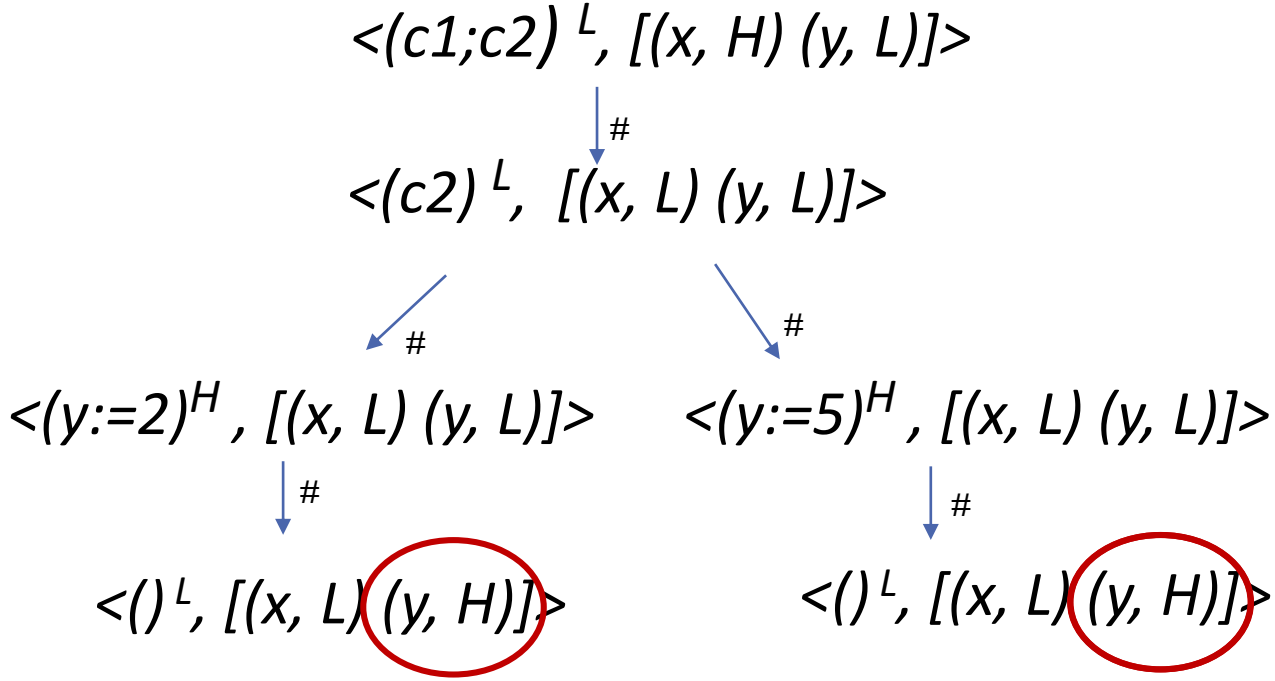
Let x:H, y:L

$M^{\#} = [(x, H) (y, L))]$

Initial execution environment: L

P
c1: y:=7;
c2: if (x=0) y:=2; else y:=5;

Abstract transition system A(P)



Secure information flow violation

Stack-based low-level languages

Main problems:

- How data flow through the operand stacks

- Scope of the implicit flow computed using the control flow graph and the notion of immediate postdominator (ipd)- the first instruction common to all the branches

Java Bytecode

op	pop two operands off the stack, perform the operation, and push the result onto the stack
pop	discard the top value from the stack
push k	push the constant k onto the stack
load x	push the value of variable x onto the stack
store x	pop off the stack and store the value into x
if j	pop off the stack and jump to j if non-zero
goto j	jump to j
jsr j	at address p, jump to address j and push p+1 onto the operand stack
ret x	jump to the address stored in x
halt	stop

Standard Operational semantics

x: 5
y: 1

state: <program counter, memory, operand stack>

$$\langle PC, [MEM(x)\ MEM(y)], STACK \rangle$$

```
0 load y
1 if 4
2 push 1
3 goto 5
4 push 0
5 store x
6 halt
```

$$\langle 0, [5,1], \lambda \rangle$$
$$\downarrow_{\textbf{load}}$$
$$\langle 1, [5,1], 1 \rangle$$
$$\downarrow_{\textbf{if}_{\textbf{true}}}$$
$$\langle 4, [5,1], \lambda \rangle$$
$$\downarrow_{\textbf{push}}$$
$$\langle 5, [5,1], 0 \rangle$$
$$\downarrow_{\textbf{store}}$$
$$\langle 6, [0,1], \lambda \rangle$$

Concrete Operational semantics

x: (5,L)
y: (1,H)

ipd: immediate post-dominator
ipd(1) = 5

state: <env, program counter, memory, operand stack, ipd stack>

```
0 load y
1 if 4
2 push 1
3 goto 5
4 push 0
5 store x
6 halt
```

$$\langle ENV, PC, [MEM(x)\ MEM(y)], STACK, IPD\rangle$$

$$\langle l, 0, [(5,l),(1,h)], \lambda, \lambda\rangle$$
$$\downarrow_{\textbf{load}}$$
$$\langle l, 1, [(5,l),(1,h)], (1,h), \lambda\rangle$$
$$\downarrow_{\textbf{if}_{\textbf{true}}}$$
$$\langle h, 4, [(5,l),(1,h)], \lambda, (5,l)\rangle$$
$$\downarrow_{\textbf{push}}$$
$$\langle h, 5, [(5,l),(1,h)], (0,h), (5,l)\rangle$$
$$\downarrow_{\textbf{ipd}}$$
$$\langle l, 5, [(5,l),(1,h)], (0,h), \lambda\rangle$$
$$\downarrow_{\textbf{store}}$$
$$\langle l, 6, [(0,h),(1,h)], \lambda, \lambda\rangle$$

Abstract Operational semantics

```
0 load y
1 if 4
2 push 1
3 goto 5
4 push 0
5 store x
6 halt
```

$$\langle ENV, PC, [MEM^\natural(x)\ MEM^\natural(y)], STACK^\natural, IPD\rangle$$

$$\langle l, 0, [l,h], \lambda, \lambda\rangle$$

$$\downarrow \textbf{load}$$

$$\langle l, 1, [l,h], h, \lambda\rangle$$

$$\swarrow \textbf{if}_{\textbf{true}} \qquad \textbf{if}_{\textbf{false}} \searrow$$

$$\langle h, 4, [l,h], \lambda, (5,l)\rangle \qquad \langle h, 2, [l,h], \lambda, (5,l)\rangle$$

$$\downarrow \textbf{push}$$

$$\downarrow \textbf{push} \qquad \langle h, 3, [l,h], h, (5,l)\rangle$$

$$\downarrow \textbf{goto}$$

$$\langle h, 5, [l,h], h, (5,l)\rangle$$

$$\downarrow \textbf{ipd}$$

$$\langle l, 5, [l,h], h, \lambda\rangle$$

$$\downarrow \textbf{store}$$

$$\langle l, 6, [h,h], \lambda, \lambda\rangle$$

Information Flow in programs

UNIVERSITÀ DI PISA

- Information flow occurs through
  - simple variables, input/output files
  - array, structures, objects
  - pointers, references
  - global variables
  - function calls
    (parameters by value, parameters by reference, return)

Function invocations

If a function call is executed in the scope of a conditional instruction, the function is executed under the implicit flow.

```
if (y < 0)
    then   f();
```

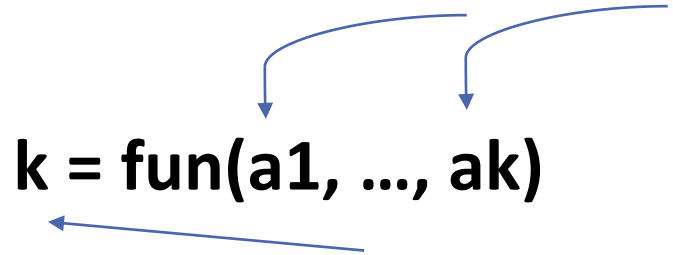
Function f() is invoked depending on the value of variable y.

Instructions of f() are executed under the implicit flow of the condition of the if statement

Function invocation

Data propagation caused by actual parameter and return of a function

```
type fun (type x1, …,  type xn) {
        ……..
    return expr;
}
```



The security context

Secure information flow studied by using a security context

- For each global variable: the highest level of data stored

  var: σ

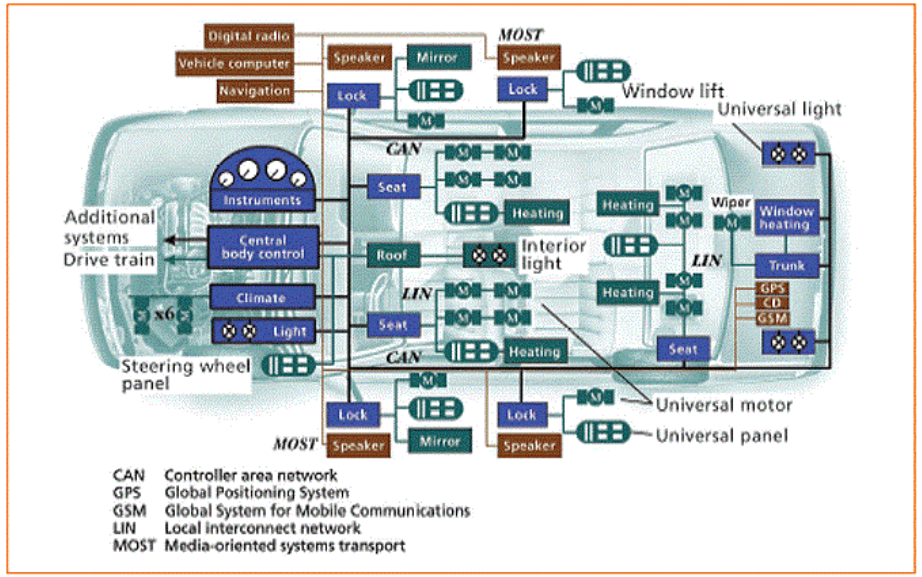
- For each function: the highest level of input/output parameters, return and the security environment of each invocation

fun(σ1, …, σn): σ, σ'

return calling environment

A case study: secure flow in AUTOSAR models

Modern automotive electronics systems are real-time embedded system running over networked Electronic Control Units (ECUs) interconnected by wired networks such as the Controller AreaNetwork (CAN) or Ethernet.



Over 80 different embedded processors, interconnected with each other.

Key ECUs (Electronic Control Unit):

- Engine Control Modul (ECM)
- Electronic Brake Control Module (EBCM)
- Transmission Control Module (TCM)
- Vehicle Vision System (VVS)
- Navigation Control Module (NCM)
- …

A case study: secure flow in AUTOSAR models

Automotive systems: Mixed-criticality safety critical systems

High criticality
Braking system, Throttle system, …

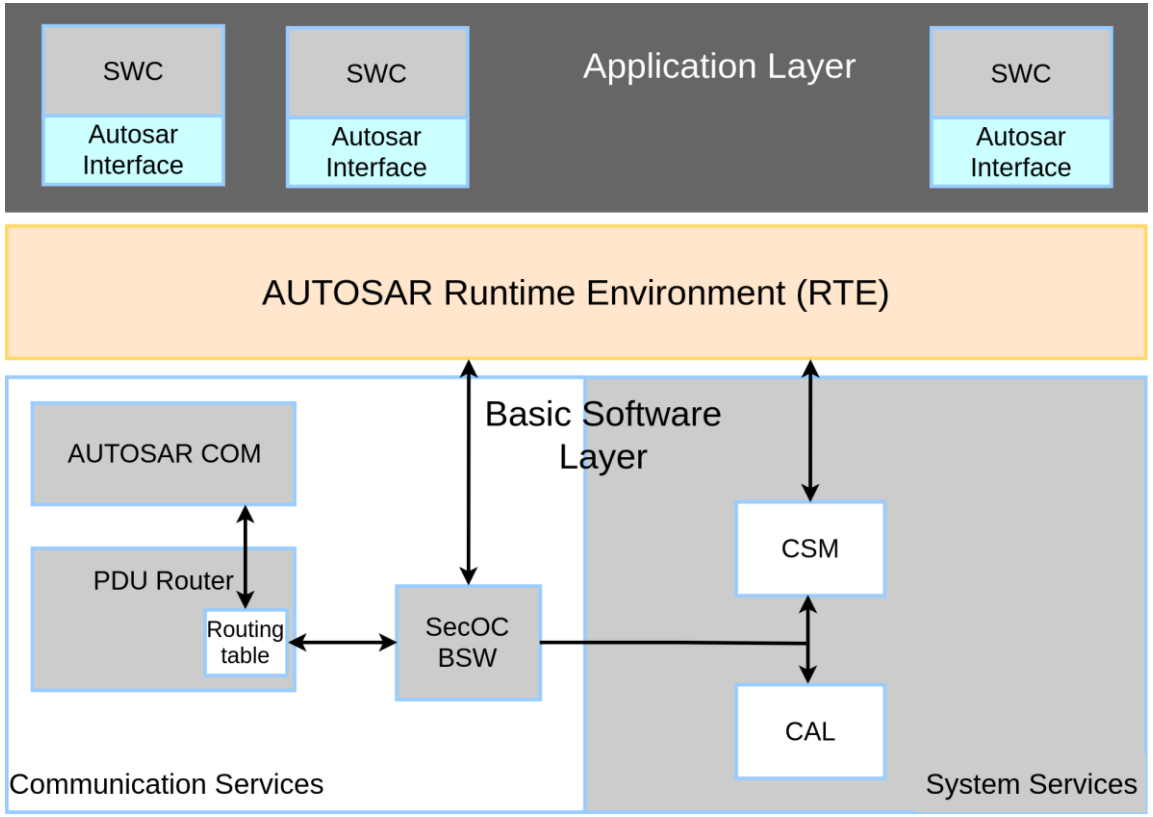
Low criticality
Infotainment system, ..

Recent research has shown that it is possible for external intruders to compromise the proper operation of safety functions getting access to the infotainment system.

Low security level data must not compromise the computation of high criticality functions

A case study: secure flow in AUTOSAR models

AUTomotive **O**pen **S**ystems **AR**chitecture: open industry standard for automotive software architectures, spanning all levels, from device drivers, to operating system, communication abstraction layers and the specification of application-level components



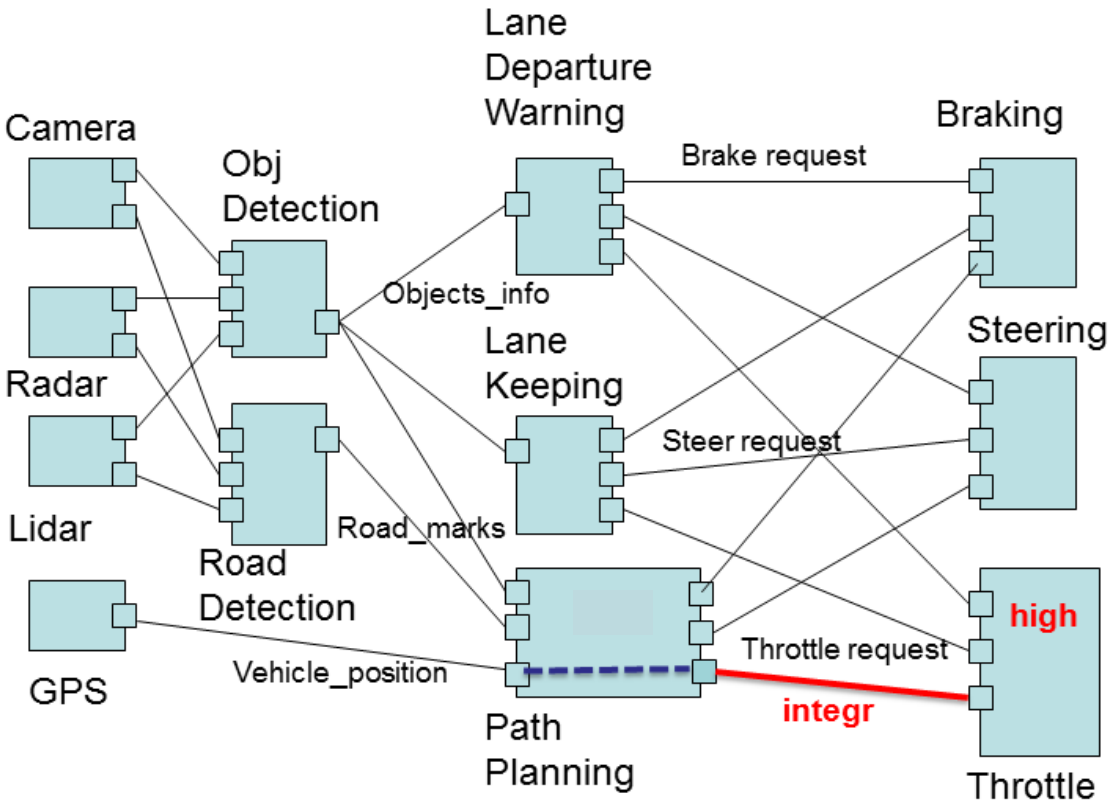
Mixed-criticality

Path Planning, Lane Keeping and Lane Departure Warning are active safety functions that receive such data and send commands to actuators (steering, throttle and brakes).

- Throttle component is assigned the high trust level;

- Throttle request link is assigned the integrity security requirement.

Autonomous driving



AUTOSAR models are extended with security annotations.

Mixed-criticality

Data received by Throttle on the link Throttle_request must satisfy high trust level and integrity security requirement

The point is that:
the way in which security annotations are specified must consider the causal dependencies between data that traverse the model.

If Throttle requires integrity on its input data sent by Path Planning,
then integrity must be guaranteed also along the path from the data originator (GPS) to Path Planning (the Vehicle_position link),
otherwise, the security constraint cannot be satisfied and the set of annotations is not correct.

Similarly, Path Planning and GPS must have high trust level.

AUTOSAR security annotations

The simplest solution assigns integrity/high to all links/components directly or indirectly connected to Throttle/Throttle_request.

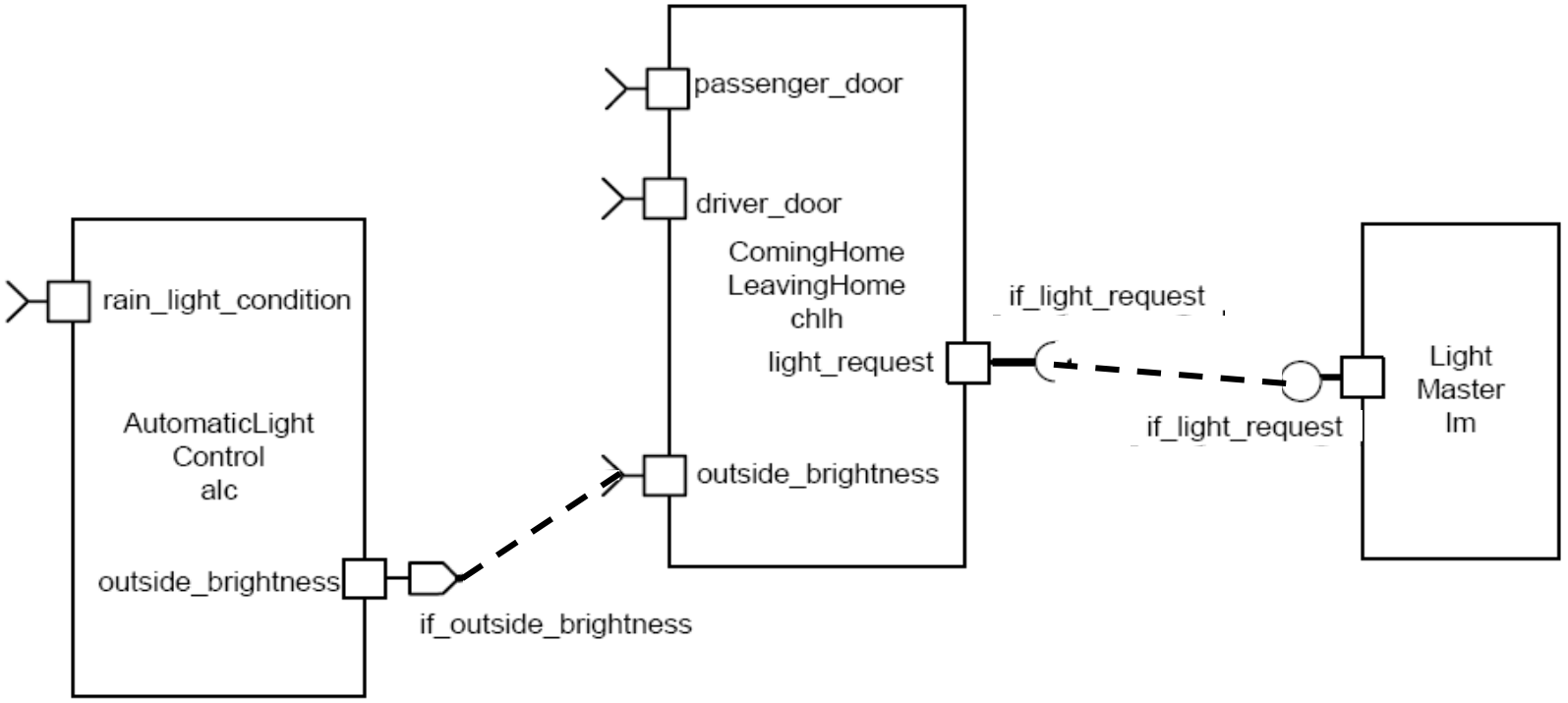
In order to obtain a more efficient solution, information flow theory can be exploited to compute the dependency between data

AUTOSAR architecture

A fundamental concept of AUTOSAR is the separation between:

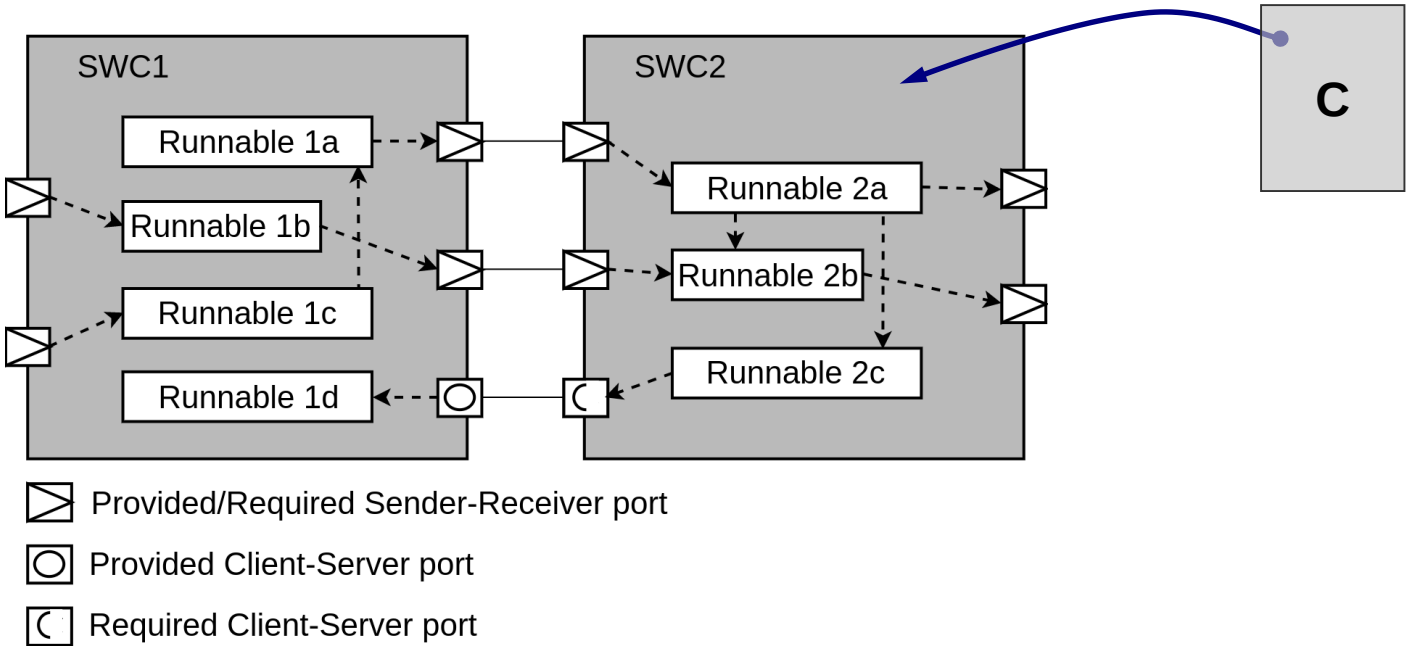
- **application** and
- **infrastructure**.

An application in AUTOSAR consists of Software Components interconnected by connectors



Runnables

- Runnables define the behavior of components
- Runnables are entry points to code-fragments and are (indirectly) a subject for scheduling by the operating system.



AUTOSAR runnable interaction

Runnable interaction

Global variables

Ports define interaction points between (runnables belonging to) different SWCs.

For interactions among runnables belonging to the same component Inter Runnable Variables (IRVs)

The RTE provides protection mechanisms for IRVs (as opposed to global variables)

AUTOSAR security policy

UNIVERSITÀ DI PISA

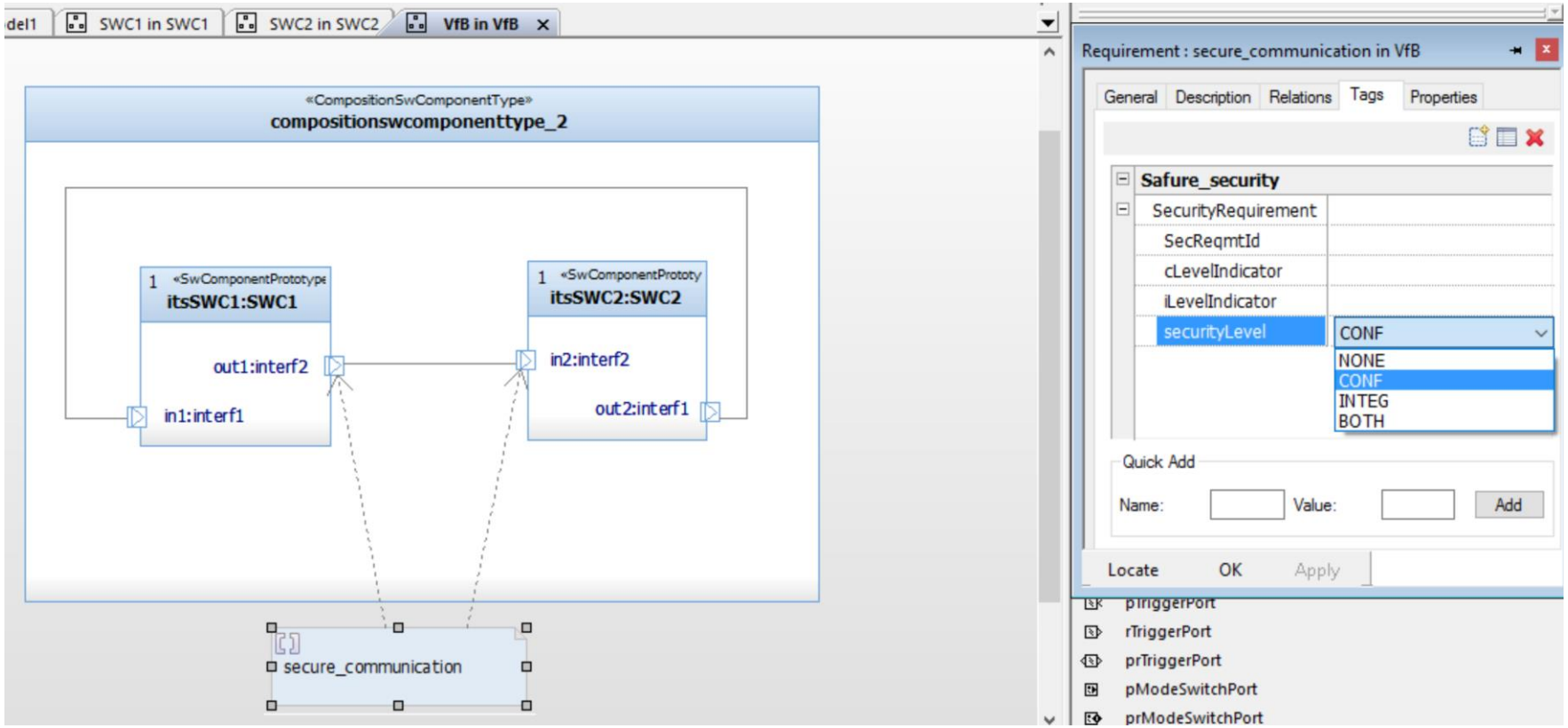
- **Trust level** of a software component
  software components with high trust level are executed on secure and reliable hardware
  -we assume two trust levels: **high, low**

- **Security requirement** of a communication link
  the level of security that data sent on links must satisfy to protect in-vehicle communications from cyber threats such as eavesdropping, integrity and spoofing.

The proposed security extensions are:
confidentiality and integrity of the exchanged information

-the security requirement can assume one of the following values:
none, conf, integr, both.

AUTOSAR extensions in Rhapsody



AUTOSAR Secure Flow analysis

An AUTOSAR model satisfies data secure flow if data sent on a link at run-time, always have a security requirement and a trust level not lower than those specified by the security annotations.

For each link, we compute:

- the lowest trust level of data sent on the link

- the lowest security requirement of data sent on the link

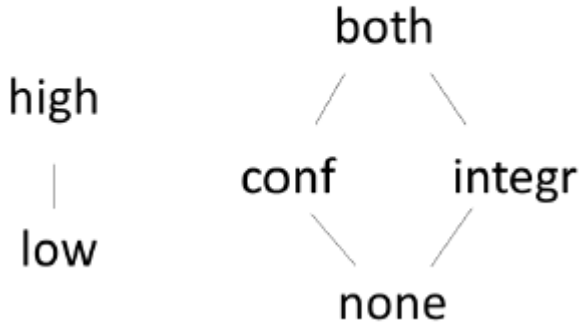
AUTOSAR Secure Flow analysis

Deps(p); set of ports
on which data sent
at port p depends

Abstract interpretation

Information flow analysis

Lattice of security levels



glb: greatest lower bound between levels
lub: least upper bound between levels

The abstract interpreter: EXEC

Each runnable is executed starting from the abstract memory and the context file, and applying the abstract rules.

All branches of conditional/iterative instructions are always executed, due to the loss of real data in the abstract semantics

Abstract semantics

A PORT is a variable.

RTE function for reading from or writing onto ports are mapped to read and write of the port variable.
For simplicity, the name of the port variable is equal to the name of the port.

RTE functions that invoke remote services trigger the runnable that implements the service. The function implementing the service is invoked

Abstract semantics

A POINTER is assumed to be simple variable, that maintains the dependencies of the pointer, plus the dependencies of the pointed data in the abstract execution.

An ARRAY is assumed to be a simple variable, that maintains the whole dependencies of each element in the array.

A STRUCTURED VARIABLE is mapped to a set of simple variables, one for each member (we use the notation, as usual). If we have a variable data that is a structure with two fields a and b, we map such variable into two simple variables, data:a and data:b, respectively.

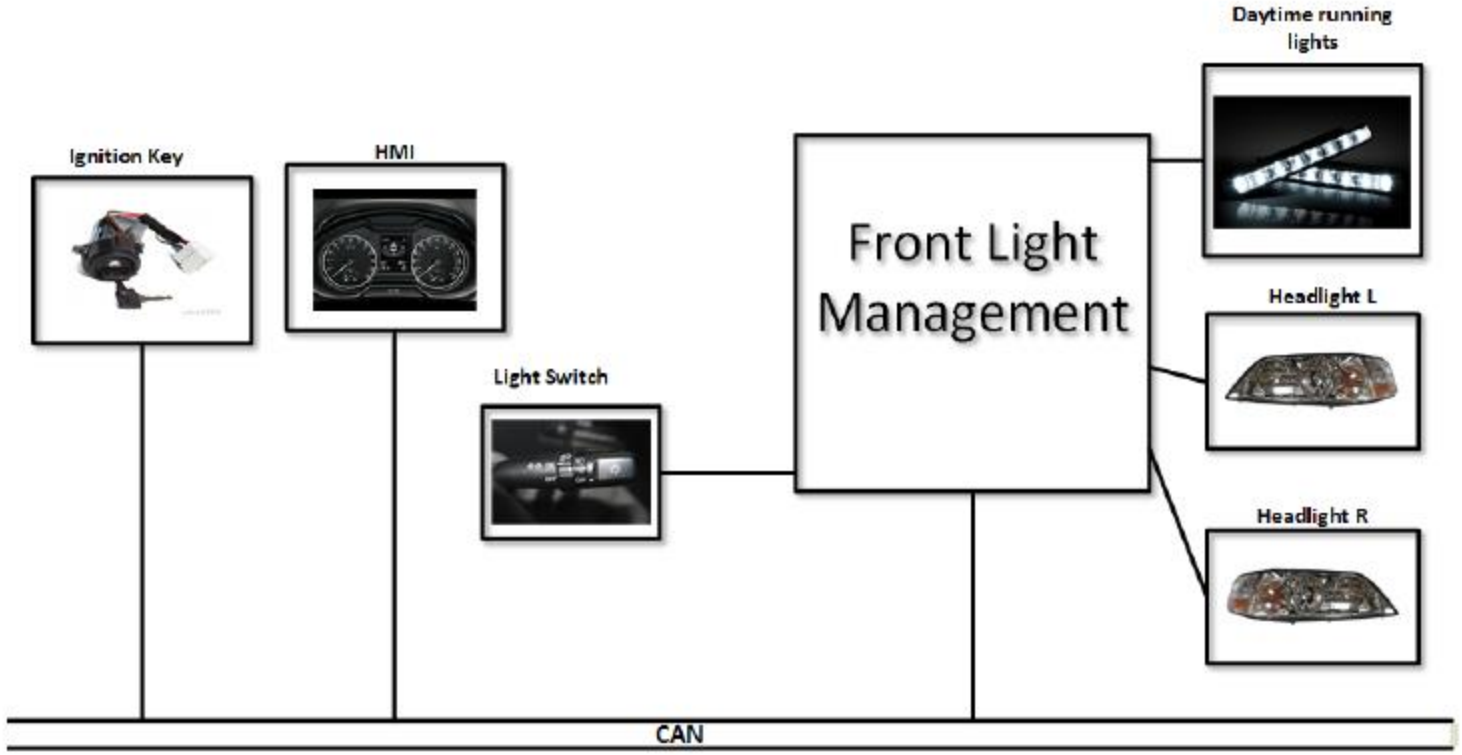
Iterative analysis

Iterative analysis until fixpoint is reached

A: security context
R: set of all runnables

$$
\begin{aligned}
&A := A^{\emptyset} \\
&T := R \\
&\texttt{while}(T \neq \emptyset) \\
&\quad \texttt{select}\ r \in T \\
&\quad T := T - \{r\} \\
&\quad A' := EXEC(r, A) \\
&\quad \texttt{if}(A' \neq A) \\
&\quad\quad A := A' \\
&\quad\quad T := R
\end{aligned}
$$

An example: Front Light Manager



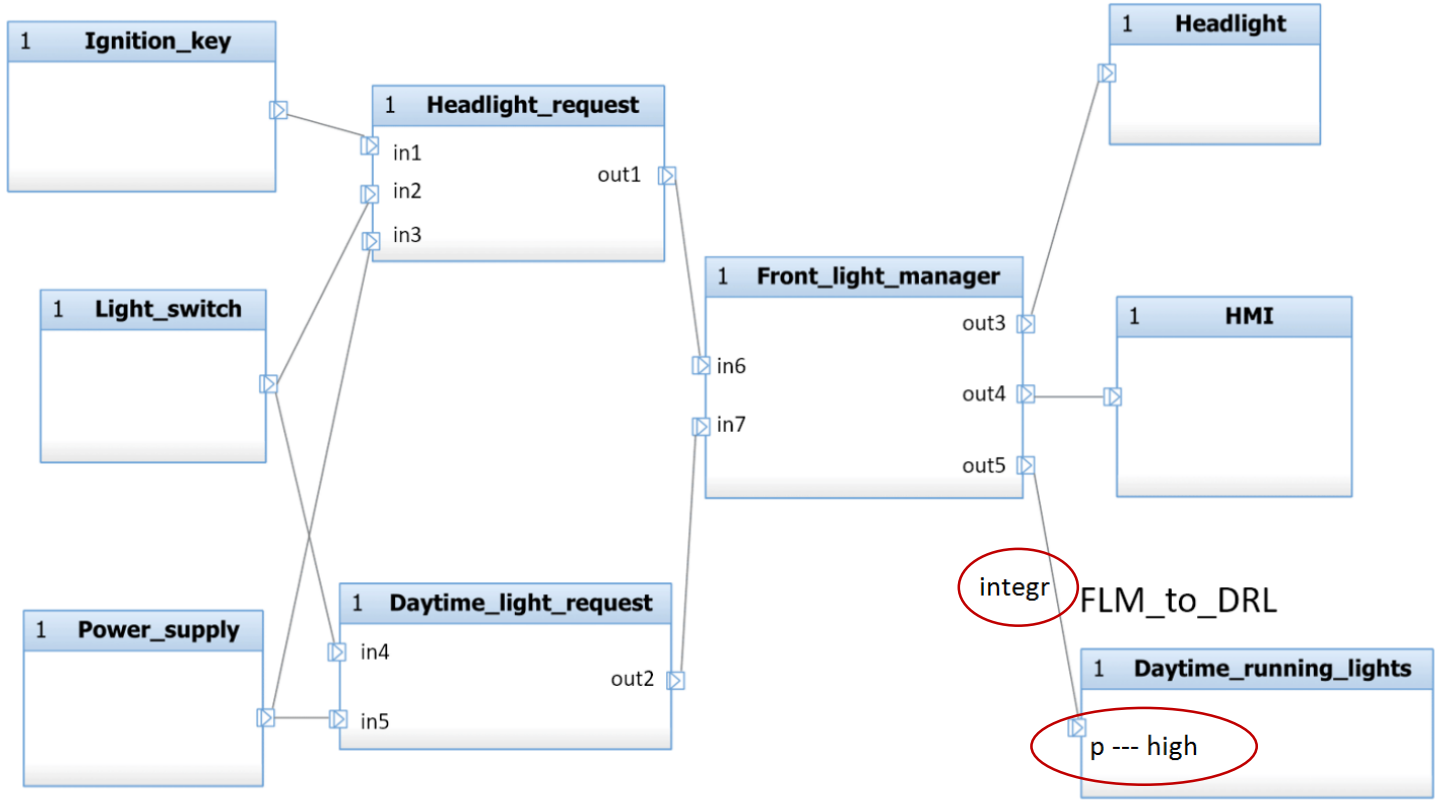
Safety Use Case Example, release 4.2.2. http://www.autosar.org/fileadmin/files/releases/4-2/software-architecture/safety-and security/auxiliary/AUTOSAR_EXP_SafetyUseCase.pdf

Front Light Manager

Security annotations: Daytime_running_lights : High FLM_TO_DRL : integr

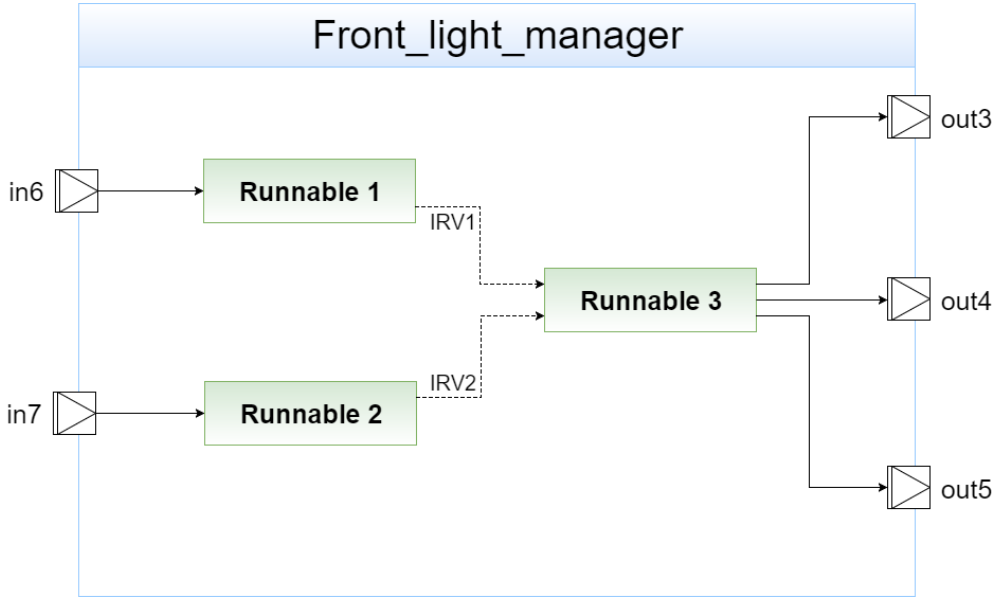
Data secure flow
is not satisfied

data sent on the
link FLM_TO_DRL
are not protected
along the path
from the sources to
the destination



Simplest solution: assignment of high trust level to Front_light_manager, Headlight_request, Daytime_light_request, Light_switch, Ignition_key, Power_supply. Similarly for links.

An example of component: Front_light_manager




```
void FLM_Runnable3(void) {
    Rte_IWrite_Runnable3_PPort_out3(Rte_IrvIRead_Runnable3_IRV1());
    Rte_IWrite_Runnable3_PPort_out5((Rte_IrvIRead_Runnable3_IRV2());
    if ( (Rte_IrvIRead_Runnable3_IRV1() == REQ_HEADLIGHT_ON) ||
         (Rte_IrvIRead_Runnable3_IRV2() == REQ_DAYTIME_ON) ){
              Rte_IWrite_Runnable3_PPort_out4(LIGHTS_ON);
    }
    else{
          Rte_IWrite_Runnable3_PPort_out4(LIGHTS_OFF);
    }
}
```

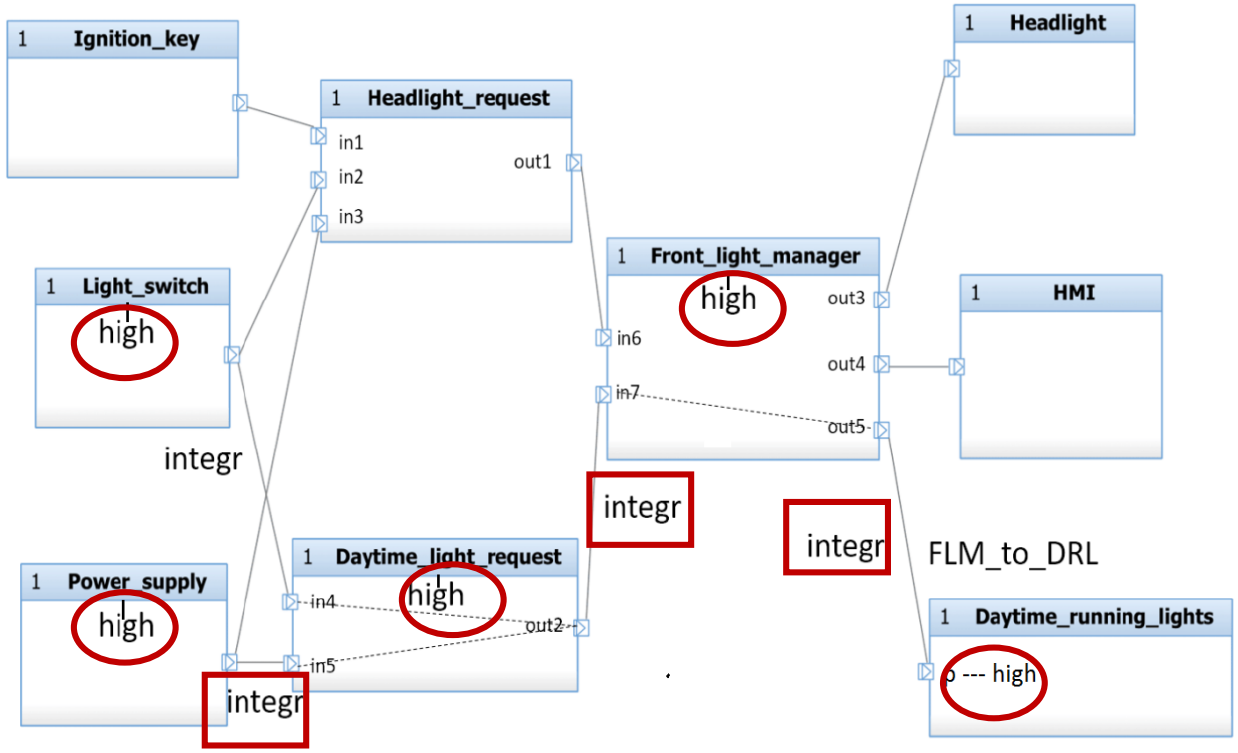
Information for generating the context

```
% global variables
int HR_voltage_threshold1;
int HR_voltage_threshold2;
int DLR_voltage_threshold1;
...
% inter runnable variables
int16_t FLM_IRV1;
int16_t FLM_IRV2;
int16_t DLR_IRV1;
...
```

```
% ports
int in1;
int in2;
...
int out1;
int out2;
...
% functions
void flm_Runnable1() 0;
void flm_Runnable2() 0;
.....
% links
out2 -> in7;
out1 -> in6;
```

Using Deps to annotate the model

the output port of Front light manager connected to the Daytime_running_lights (out5 in our implementation) does not depend on the input port connected to the Headlight request component (in6 in our implementation)



Model satisfies Secure flow property

Conclusions

Abstract interpretation allows automated verification of secure information flow in programs

Intermediate level between typing approaches and sematics-based approaches

Analysis can be improved to reduce the number of false positive

Other works

- Secure information flow in Java cards applications
- Secure information flow in concurrent programs

Conclusions

Future work

- Privacy of data in Android smart phones
- Malicious Colluding apps
- Privacy of data in medical app

References

R. Barbuti, C. Bernardeschi, N. De Francesco, Abstract interpretation of operational semantics for secure information flow, Information Processing Letters, num. 2, vol. 83, pp. 101-108, 2002.

R. Barbuti, C. Bernardeschi, N. De Francesco, Analyzing Information Flow Properties in Assembly Code by Abstract Interpretation, Computer Journal, num. 1, vol. 47, pp. 25-45, 2004.

Avvenuti M, Bernardeschi C, De Francesco N, Masci P. JCSI: A Tool for Checking Secure Information Flow in Java Card Applications . The Journal of Systems and Software, vol. 85, p. 2479-2493, 2012.

Cinzia Bernardeschi, Marco Di Natale, Gianluca Dini, Maurizio Palmieri: Verifying data secure flow in AUTOSAR models. J. Computer Virology and Hacking Techniques 14(4): 269-289, 2018.