Verifying Data Secure Flow in AUTOSAR Models by Static Analysis

Cinzia Bernardeschi[1], Marco Di Natale[2], Gianluca Dini[1] and Maurizio Palmieri[1]
[1]*Department of Information Engineering, University of Pisa, Largo L. Lazzarino 1, Pisa, Italy*
[2]*Scuola Superiore Sant'Anna, Piazza Martiri della Libertà 33, Pisa, Italy*

Keywords: AUTOSAR, Security, Information Flow, Static Analysis.

Abstract: This paper presents a method to check data secure flow in security annotated AUTOSAR models. The approach is based on information flow analysis and abstract interpretation. The analysis computes the lowest security level of data sent on a communication, according to the annotations in the model and the code of runnables. An abstract interpreter executes runnables on abstract domains that abstract from real values and consider only data dependency levels. Data secure flow is verified if data sent on a communication always satisfy the security annotation in the model. The work has been developed in the EU project Safure, where modeling extensions to AUTOSAR have been proposed to improve security in automotive communications.

1 INTRODUCTION

Modern automotive electronics systems are real-time embedded system running over networked Electronic Control Units (ECU) interconnected by wired networks such as the Controller AreaNetwork (CAN) or Ethernet. Moreover, wireless connectivity is increasingly used for additional exibility and bandwidth for features like keyless entry, diagnostic, and entertainment.

Recent research has shown that it is possible for external intruders to intentionally compromise the proper operation and functionality of these systems. Koscher et al. demonstrated that if an adversary were able to communicate on one or more of a car internal network buses, then this capability could be sufficient to maliciously control critical components across the entire car (including dangerous behavior such as forcibly engaging or disengaging individual brakes independent of driver input) (Koscher et al., 2010). These results raise the question of whether and how an adversary might be able to access a car internal bus (and thus compromise its ECUs) in the case of absent direct physical access. Checkoway et al. demonstrated that external attacks are indeed feasible (Checkoway et al., 2011) and categorized external attack vectors as a function of the attacker ability to deliver malicious input via particular modalities: indirect physical access, shortrange wireless access, and long-range wireless access. Charlie Miller and Chris Valasek have recently demonstrated further remote attacks (Wyglinski et al., 2013).

Recently, many research and industrial activities have started to take security into account in the early phases of the development cycle of automotive electronics systems, both by enforcing software programming standards that prevent software defects that may enable cyber-attacks (Checkoway et al., 2011), as well as by implementing security mechanisms for secure communication (Lemke et al., 2006) including software delivery, installation and flashing (Adelsbach et al., 2006)(Stephan et al., 2006).

The AUTomotive Open System ARchitecture (AUTOSAR) standard is the standard for the modeling and development of software components in the automotive industry (AUTOSAR, a). In AUTOSAR, safety and security services are being standardised with respect to the set of basic services that may be required by the application, such as the basic cryptographic functionalities provided by the Crypto Service Manager (CSM) or the definition of integrity-related message authentication codes (MACs) in messages (SecOC component). These modules are not currently matched by corresponding models for security at the application level.

The work (Bernardeschi et al., 2016) aims at bridging this gap. A set of modelling extensions to address cybersecurity requirements at modelling stage in AUTOSAR have been defined, and a code generation tool has been developed that automatically synthesizes the right services to use to achieve the security level specified by the developers. Security re-

Bernardeschi, C., Natale, M., Dini, G. and Palmieri, M.
Verifying Data Secure Flow in AUTOSAR Models by Static Analysis.
DOI: 10.5220/0006288707040713
In *Proceedings of the 3rd International Conference on Information Systems Security and Privacy (ICISSP 2017)*, pages 704-713
ISBN: 978-989-758-209-7

quirements are assigned to system components and in-vehicle communication links between components, and they are realized as stereotypes extending the AUTOSAR implementation provided by the IBM Rhapsody tool.

However, the way in which security annotations can be specified, does not consider the problem of dependencies between data that traverse components and communication links in the AUTOSAR model. For example, if integrity is requested for input data to the Brake actuation sub-system, all the communication links traversed from the sensors originating the data to the Brake sub-system must be protected. Otherwise, the security constraint cannot be satisfied. We introduce the concept of *data secure flow*. Data secure flow is verified if, for every possible execution, data sent on a communication always satisfy the security annotation written in the model.

Dependencies between data in an AUTOSAR model, can be studied using approaches for checking secure information flow in programs (D. E. Denning, 1977). In particular, we use an approach based on abstract interpretation (Cousot and Cousot., 1992), a static analysis technique for the automatic extraction of information about the possible executions of computer programs. Abstract interpretation has been used in (Barbuti et al., 2002) to analyse secure information flow in a simple imperative language. We extend the technique to cover the analysis of AUTOSAR models. The main points of the approach are:

- an abstract interpreter executes the functional units of software components on abstract domains that abstract from real values and consider only data dependency levels. A fixpoint iterative analysis computes the dependency between data written/read at ports of the software components.
- since the analysis computes all possible dependencies for any real execution of the functional units, the lowest security level of data sent on a communication is counted.
- data secure flow property is satisfied if the security level computed by the analysis for data sent on a communication always satisfy the security annotations in the model.

We reduce the complexity of the analysis by using static program analysis techniques (Nielson et al., 2005), which analyse the source code without executing the program. Static analysis techniques are applied for enforcing information flow security in programs in several works, the reader can refer to (Sabelfeld and Mayers, 2003) for a survey. One of the advantages of our approach is that, being based on abstract intepretation, the analysis can be fully automated. Moreover, the analysis scales up, since AUTOSAR software components are analysed separately.

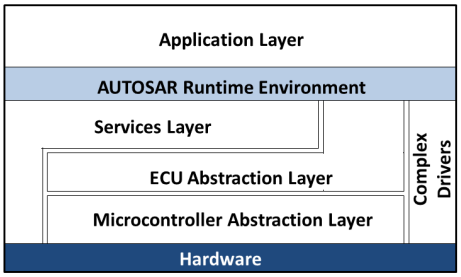

Figure 1: AUTOSAR architecture.

The paper is organized as follows: Section 2 briefly introduces AUTOSAR and security annotated AUTOSAR models. Section 3 outlines the data secure flow problem addressed by this work. Section 4 introduces the problem of dependency between data read/written at ports of software components in an AUTOSAR model. Section 5 presents the secure flow verification technique by assuming data dependencies already determined. The method used for computing data dependencies at the ports is shown in Section 6. Section 7 concludes the paper.

2 AUTOSAR

AUTOSAR (AUTOSAR, a) is an open industry standard for automotive software architecture, founded in 2003 and developed by a partnership of automotive Original Equipment Manufacturers (OEMs), suppliers and tool vendors. AUTOSAR provides a standard language for the description of application components and their interfaces; and a methodology for the development process.

A fundamental concept in AUTOSAR is the separation between application and infrastructure, see Figure 1. In particular, AUTOSAR defines a three-layered architecture consisting of:

- Application layer
- Runtime Environment (RTE) layer
- Basic Software (BSW) layer

The application layer contains the Software Components (SWCs) developed for the automotive system functions by suppliers. The RTE layer is a middleware layer, automatically generated by tools and providing a communication abstraction for software components. Finally, the BSW provides basic services and basic software modules to software components.

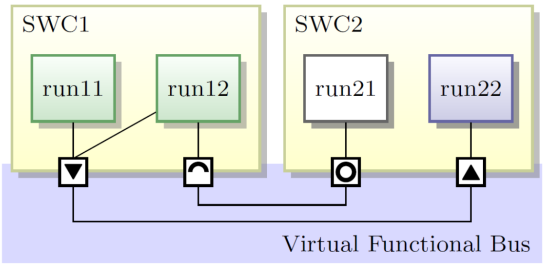

Figure 2: An example of AUTOSAR application model.

The application SWCs communicate using ports that express client-server relationships (in this case the port is typed by an operation interface) or send-receive data interactions (in this case the port is typed by a data interface consisting of a set of typed data items). The development of the SWCs is based on the Virtual Functional Bus (VFB) specified by AUTOSAR to deliver the conceptual foundation for the communication of SWCs with each other and the use of BSW services.

The internal behavior of SWCs consists of runnables or functional units, represented by a function entry point. Each runnable indicates the port it uses. Internally, the runnable code accesses the ports through a set of standard API for port communication and port service request (denoted as RTE services). A set of events triggering the execution of runnables completes the set of the main modeling entities. A runnable can be triggered by a timing event, by a data send event, a data receive event or by the invocation of a server call at the server port.

An example is shown in Figure 2. Runnable `r11` of SWC1 communicates with runnable `r22` of SWC2 through sender-receiver ports; Runnable `r12` of SWC1 communicates with with runnable `r21` of SWC2 through client-server ports. Moreover, `r21` is triggered by the invocation of the service at the server port.

AUTOSAR allows software components to be developed independently of the underlying hardware, which means that they are transferable and reusable.

2.1 Security Annotated Autosar Models

In (Bernardeschi et al., 2016), AUTOSAR models are extended with *security annotations*. Two modelling extensions are introduced:

- the *trust level* of a software component
- the *security requirement* of a communication link

A software component may be associated to a trust level which specifies to what extent the element can be trusted to provide the expected function, or service with respect to attacks targeted to compromise the functionality of the element. Without loss of generality, we assume two trust levels: *high* and *low*. Software components with high trust level are executed on secure and reliable hardware.

To protect in-vehicle communications from cyber threats such as eavesdropping, integrity and spoofing, a communication link may be associated to a security requirement which represents the level of security that data sent on the link must satisfy. 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*. These four values codify no security, confidentiality, integrity and, both confidentiality and integrity, respectively.

Figure 3 shows an example of AUTOSAR annotated model. The example represents an application in which data collected by sensors (lidars, radars and cameras), together with the position information coming from the GPS system, are used to detect road-markings and objects (pedestrian, vehicles) on the road. 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).

For semplicity, we assume that 1) Throttle and PathPlanning software components are assigned high trust level; 2) the other components are assigned low. Moreover, we assume that 1) Throttle request link is annotated with data integrity security requirement; 2) the other comminication links have no security requirements.

Finally, we assume send/receive data communications between components. Moreover, in the figure, dependencies between data read/writen at ports of PathPlanning are shown as dotted lines internal to the component.

3 PROBLEM STATEMENT: DATA SECURE FLOW PROPERTY

We introduce the following definitions:

- *Data Security*. Every data is assigned a pair ⟨`trust level`,`security requirement`⟩ that characterises its degree of security. As data flow through the components and the communication links, its *data security* is updated.
- *Data Secure Flow Property*. When the system is in operation, *data security* of data sent on a link must have no lower trust level than the level of the

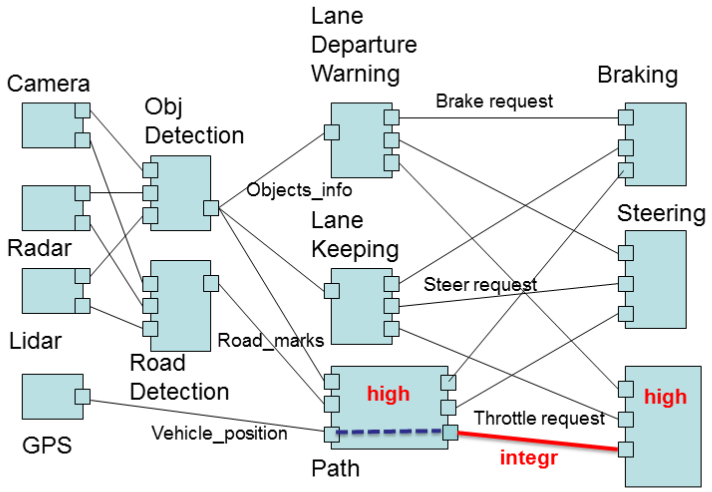

Figure 3: An example of security annotated model.

receiver component and no lower security requirement than the requirement of the link, both fixed at design time through the security annotations in the AUTOSAR model.

With reference to Figure 3, the trust level of data received by Throttle component must be high. Throttle receives data directly by PathPlanning, a high trust level component, which in turn, receives data GPS, which has a low trust level. Therefore *Data secure flow* is violated. Assume GPS has a high trust level. Data sent on Throttle request link must satisfy integrity requirement. The dependency between data read/written at ports in PathPlanning, makes the property false, because such data depends on Vehicle_position link, which does not have any security requirement.

The problem is that security annotations are local to single components and links, and the *data secure flow* property may or may not be satisfied due to data dependencies in the model.

What we propose is a static analysis approach to check *data secure flow* property in security annotated models. For the analysis, we define an ordering between security values with the meaning that if security value s preceedes security value s', then s is in some way "lower in security degree" to s'.

Fundamentals to the analysis are the structure of the system model (components, runnables, ports and links), the ordering relation mentioned above, and the dependencies between data in the model.

4 DATA DEPENDENCIES IN AN AUTOSAR MODEL

The basic idea is modelling ports as variables, and modelling runnables as functions in the programming language. In particular,

- for send-receive data communications, reading a data at a port is equivalent to reading the variable; writing a data at a port is equivalent to writing the variable.
- for client-server communications, the client request is equivalent to a function call, that corresponds to the invocation of the runnable implementing the requested service.
- for updates of variables that trigger the execution of runnables, the write of the variables corresponds to the invocation of the functions implementing the runnables. If a variable corresponds to a port of a send/receive data communication, a write of the variable may trigger a runnable local to the sender SWC or a runnable at the receiver SWC.

The analysis we present is based on abstract interpretation technique (Cousot and Cousot., 1992):

- the standard operational semantics of the programming language is enhanced to include information useful in the analysis.
- abstract domains are identified and abstract semantics rules are defined that execute the program on abstract domanins.
- the abstract rules compute the flow of information in the program

The rules take into account explicit and implicit flow of information. Let v_{pi} and v_{pj} be variables correspondent to p_i and p_j ports.
An example of explicit information flow is:

$$x := v_{pi}; \quad v_{pj} := x + 3;$$

An example of implicit information flow is:

$$\texttt{if } (v_{pi} == -1) \texttt{ then } v_{pj} := 0 \texttt{ else } v_{pj} := 1$$

In both the examples above, the value of variable v_{pj} depends on the value of variable v_{pi}. Data written to port p_j depends on data read at port p_i.

For functions, information flows through function parameters and return. An example of explicit information flow is:

$$x := v_{pi}; \quad f(x);$$

The call to a function without parameters and return, can be the cause of an information flow, in case of implicit flow:

$$\texttt{if } (v_{pi} == -1) \texttt{ then } f();$$

Function f is invoked depending on the value read at port p_i.

5 DATA SECURE FLOW PROPERTY VERIFICATION

Given an AUTOSAR model, we use the following notations and definitions:

- $C = \{c_1, c_2, \cdots, c_k\}$ is the set of SWCs.
- $P = \{p_1, \cdots, p_n\}$ is the set of ports of the SWCs.
- $L = \{l_1, \cdots, l_m\}$ is the set of links. A link denotes a connection between two ports. The link $l = (p_i, p_j)$ connects the port p_i to the port p_j, with p_i output port of the sender SWC and p_j input port of the receiver SWC.
- $\texttt{trustlevel}(c)$ is the trust level assigned to the software component c.
- $\texttt{securityrequirement}(l)$ is the security requirement assigned to link l.

In addition we use the following definitions and functions:

- Given a port p, $cmp(p)$ is the component to which the port belongs.
- Given a port p, $Deps(p)$ is the set of ports on which the data written at port p depends (see section 6).

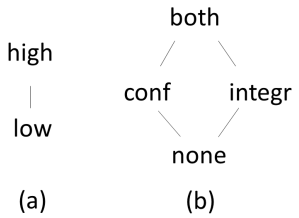

Figure 4: Order relation on (a) trust levels (b) security requirements.

Given a link $l = (p_i, p_j) \in L$,
1. $\langle \delta_l, \mu_l \rangle = \langle high, both \rangle$
2. $\forall p \in Deps(p_i)$
   $\delta_l = glb(\delta_l, \texttt{trustlevel}(cmp(p)))$
3. $\forall l' = (q, q'), \mid q, q' \in Deps(p)$
   $\mu_l = glb(\mu_l, \texttt{securityrequirement}(l'))$

Figure 5: Algorithm for data security of link l.

Definition 1 (Trust Level). *Let $A = \{low, high\}$ be the set of trust levels, ordered by $low \sqsubset high$, where $\sqsubset$ is the lower between levels, see Figure 4 (a). Let* glb *denote the greatest lower bound, and* lub *the least upper bound between levels: it is $glb(low, high) = low$ and $lub(low, high) = high$.*

Definition 2 (Security Requirement). *Let $B = \{conf, integr, both, none\}$ be the set of security requirements of links, partially ordered by $\sqsubset$, as shown in Figure 4 (b). Let* glb *denote the greatest lower bound, and* lub *the least upper bound between levels. $(B, \sqsubset)$ is a lattice (i.e., every pair of elements of B has both a greatest lower bound and a least upper bound). For example, $glb(integr, conf) = none$; $lub(integr, conf) = both$*

Note that *conf* and *integr* are not ordered, because one is not "lower in security degree" to the other.

5.1 The Method

In the analysis, we compute the lowest trust level and the lowest security requirement of data sent on a link l ($\langle \texttt{trust level}, \texttt{security requirement} \rangle$), with the algorithm shown in Figure 5. The algorithm records in δ_l and μ_l such levels ($\langle \delta_l, \mu_l \rangle$).

Assume $l = (p_i, p_j)$. Data sent on the link l are data written at port p_i.

First the algorithm sets δ_l equal to the greatest trust level and μ_l equal to the greatest security requirement. Then for each port p on which data sent on the link l depends ($p \in Deps(p_i)$) , δ_l is updated to consider the trust level of the SWC to which the port p

belongs: the trust level δ_l is set equal to the greatest lower bound between the currrent value and the trust level of the SWC to which port p belongs. Finally, for each link l' in the model traversed by data sent on link l (source and destination ports of l' belong to $Deps(p_i)$), μ_l is updated: the security requirement μ_l is set equal to the greatest lower bound between the currrent value and the security requirement of the link l'. Note that, at each step δ_l can only be downgraded; analogously, μ_l can only be downgraded.

An AUTOSAR model satisfies data secure flow if for each communication link, 1) the trust level of destination component of the link is not greater than the trust lowest trust level of data sent on the link, and 2) the security requirement of the communication link is not greater than the lowest security requirement of data sent on the link.

Definition 3 (Data Secure Flow Property). *Given an AUTOSAR model with security annotations, the model satisfies the data secure flow property if, for each link $l = (p_i, p_j) \in L$, with $\langle \delta_l, \mu_l \rangle$ the data security of data sent at l:*

$$\texttt{trustlevel}(cmp(p_j)) \sqsubseteq \delta_l \wedge \texttt{securityrequirement}(l) \sqsubseteq \mu_l$$

6 DEPENDENCIES BETWEEN PORTS OF AN AUTOSAR MODEL

A port p_j does not depend on port p_i if data sent at p_j are independent from data received at p_i. We formally define port dependencies as follows.

Definition 4 (Port Dependencies). *Let us consider an AUTOSAR model. A port p_j does not depend on the port p_i if: for each pair of values v_1, v_2 at p_i, with $v_1 \neq v_2$, it is: $p_j(p_1, \cdots, p_{i-1}, v_1, p_{i+1}, \cdots, p_n) = p_j(p_1, \cdots, p_{i-1}, v_2, p_{i+1}, \cdots, p_n)$ for each possible execution, where $p_j(p_1, \cdots, p_{i-1}, x, p_{i+1}, \cdots, p_n)$ is the value written on port p_j when x is read from input port p_i .*

In the analysis, we define an AUTOSAR model as a tuple $A = (R, Var, C, \Theta)$, that consists of a set R of runnables, a set Var of variables, a set C of connections (links), a set Θ of levels. In particular

- $Var = VP \cup VIR \cup VG$ is a set that consists of the set VP of port variables, the set VIR of inter-runnable variables, and the set VG of global variables of SWCs.
- R is the set of all runnables. A runnable $r \in R$ is a function $r : Var \rightarrow Var$.
- $C = \{(src, dst) \mid src \subseteq VP \wedge dst \subseteq VP\}$ is a set of connections between port variables (*src* is the source and *dst* is the destination port of the connection).
- $\Theta = \{\theta_1, \cdots, \theta_n\}$ is the set of levels, one level for each port. The level of port p_i is called θ_i.

6.1 Data Dependency Levels

We assume the following set $\Theta = \{\theta_1, \theta_2, \cdots, \theta_n\}$ of dependency levels, one for each port. We consider the powerset $\Sigma = 2^\Theta$, i.e. the set of all subset of Θ, ordered by subset inclusion. $(\Sigma, \subseteq)$ is a complete lattice (every pair of elements of Σ has both a greatest lower bound, *glb*, and a least upper bound, *lub*). The *lub* is given by the union ($\cup$) and the *glb* is given by the intersection of subsets ($\cap$). Given $X \subseteq Y$, $X \cup Y = Y$ and $X \cap Y = X$. The analysis operates over levels in Σ. The singleton set $\{\theta_i\}$ denotes a dependency from input port p_i. The set $\{\theta_i, \theta_j\}$ denotes dependency on both input ports p_i and p_j. The minimum of Σ is the empty set.

6.2 Analysis of a Runnable

In the following we describe the basic concepts of the analysis.

A program is a sequence q of commands. Let m be a memory that contains all the variables accessed by the program.

The execution of a program is a transition system obtained by executing q starting from the initial memory, by applying the rules of the operational semantics of the language. As an example, the rule for a simple expression consisting of variable x is:

$$\textbf{Expr}_{var} \quad \frac{}{\langle x, m \rangle \longrightarrow_{expr} m(x)}$$

The semantics of the exptression x is the value of x in memory m.

The rule for the assignment $\texttt{x} := \texttt{e}$ is the following, where e is an expression, and $m[k/x]$ the memory m, where the variable x is assigned the new value k:

$$\textbf{Ass} \quad \frac{\langle e, m \rangle \longrightarrow_{expr} k}{\langle x{:=}e, m \rangle \longrightarrow m[k/x]}$$

If k is the evaluation of the expression e in memory m, the semantics of $x{:=}e$ changes the memory by assigning value k to variable x.

The operational semantics of the language is extended to convey the set of data dependency levels during the execution.

- Each value is annotated with the set of dependencies (both implicit and explicit);

Data become pairs (k,τ), where k is the value and τ is the dependency level.

- and each command is executed under an environment σ that represents the *lub* of the dependencies of the open implicit flows.
  The notation $(x{:=}e)^{\sigma}$, represents the execution of the assignment under an environment σ.

For example, the dependencies of a variable expression, is given by le *lub* between the level of the data in the variable and the level of the environment in which the command is executed.

The previously introduced rules become the following, where M is the memory defined on extended values:

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

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

The abstract semantics, abstracts from actual values and maintains only annotations on data dependencies. let $M^{\sharp}$ be the abstract memory. The abstract semantics for the previous rules is following:

$$\textbf{Expr}_{var} \quad \frac{M^{\sharp}(x) = \tau}{\langle x^{\sigma}, M^{\sharp}\rangle \longrightarrow_{expr} lub(\sigma,\tau)}$$

$$\textbf{Ass} \quad \frac{\langle e^{\sigma}, M^{\sharp}\rangle \longrightarrow_{expr} \tau, \sigma' = lub(\sigma,\tau)}{\langle (x{:=}e)^{\sigma}, M^{\sharp}\rangle \longrightarrow M[\sigma'/x]}$$

A program is executed on the abstract domain starting from the abstract initial memory, and applying the abstract rules. We note that all branches of conditional/iterative instructions are always executed, due to the loss of real data in the abstract semantics.

In previous work (Barbuti et al., 2002), it is proved that the transition system built by executing q with the enhanced semantics is the same as the transition system built with the standard semantics, for a simple high level language. We extended the semantics rules to include indirections, structures, arrays and function calls (we assume Misra-C as programming language of runnables (AUTOSAR, b)).

In particular, to deal with shared memory and functions calls, a context file is introduced (similarly to the approach in (Avvenuti et al., 2012)).

Variables are scalar in the following. For structures, a variable is used for each field. For arrays, we abstract from array indexes.

6.3 Context File

A context file is defined to record information stored in the shared memory of a SWC and information about runnable calls.

The context file maintains:

- for each variable $v \in Var$, the entry $v : \sigma$, where σ is the dependency level of data assigned to v;
- for each runnable $r \in R$, the entry $r(\sigma_1,...,\sigma_k)\sigma;\sigma'$, where $\sigma_1,...,\sigma_k$ are the levels for the actual parameter, σ is the level for the result and σ' is the level of the calling environment.

In particular, during the analysis, for each variable, the context file maintains the maximum dependency level of data recorded in the variable.

For each runnable, the context file maintains how the runnable is called in terms of the maximum level of the calling environment, the actual parameters and return. We assume parameters and return of runnables, for generality.

Each port variable is initialised to the level of the port. All other variables are initially assigned the lowest level ($\emptyset$). Runnables are initially assigned the lowest level for calling environment, parameters and return.

Entries in the context file are manged in a special way. Their dependency level never decreases. An assigment to a variable in the context file, updates the dependency level of the variable in the context file to the *lub* between its current level and the level of the assigned expression. Analogously, for runnable entries.

Example. In the following, we will consider one software component of a simple example of an AUTOSAR application (MathWorks, a). The software component manages three runnable entities, 2 input ports, 4 output ports and 4 inter-runnable variables (irv1, irv2, irv3, irv4). Runnable 1 performs the sum of the value read from the input port in1 and the value from irv3 and writes the result in irv1; runnable 2 performs the subtraction between irv1 and irv2, accumulates the result and writes it on irv4 and on the output port 4; finally Runnable 3 multiplies irv4 and the value read from input port in2 and writes the result in irv2. The data sent to the other output ports (1, 2 and 3) and in irv3 oscillate between 1 and -1 and is not related to the other elements of the component.

The code of runnable r1 in the example is shown in Figure 7.

For simplicity, we use in1, in2, and out1, out2, out3, out4 as names of input/output port variables. For example, `RPort_DE1` is called in1 and `PPort_DE1` is called out1.

A runnable is abstractly executed starting from its local memory and the context file. The initial context file and the local memories for the software component are shown in Table 1.

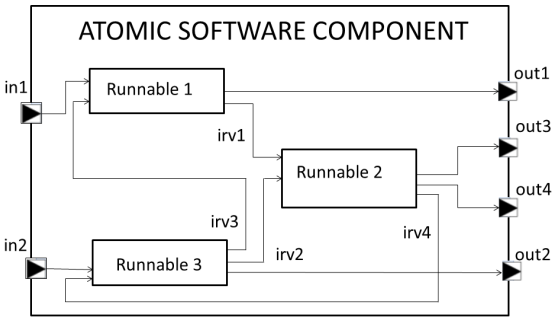

Figure 6: A software component and its runnables.

```
FUNC(void) Run1(void) {
int8_T Delay_n;
Delay_n = rtDWork.Delay_DSTATE_a;
if(((int32_T)Rte_IStatus_Run1_RPort_DE1())==0)
   {
     rtB.Add = Rte_IRead_Run1_RPort_DE1()+
    ((real_T) Rte_IrvIRead_Run1_IRV3());
   }

Rte_IrvIWrite_Run1_IRV1(rtB.Add);      (*)
rtDWork.Delay_DSTATE_a = (int8_T)((int32_T)
      (-((int32_T)rtDWork.Delay_DSTATE_a)));
Rte_IWrite_Run1_PPort_DE1(Delay_n);
}
```

Figure 7: Code of runnable r1.

6.4 Calls to RTE Functions

We deal with calls to RTE functions in the runnable code as follows.

- Send/Receive data communication ports
  RTE function for writing/reading a port are mapped to read/write of the corresponding variable:
  `Rte_IRead_Run1_RPort_DE1()` is mapped to the read of variable `RPort_DE1`
  `Rte_IWrite_Run1_PPor_DE1(Delay_n)` is mapped to the write of variable of the port: `PPort_DE1 := Delay_n`.
- Updates of variables that trigger runnables
  The calling environment of the triggered runnable is updated in the context file. The runnable is executed into an environment which depends on the level of the variable.
- Client/server ports
  RTE function for client/server communication
  This function trigger the runnable that implements the service. The calling environment of the triggered runnable is updated in the context file. The runnable implementing the service is executed into an environment which depends on the level of the environment of the caller.

$F := F_0, T := R$
while$(T \neq \emptyset)$
  select $r \in T$
  $T := T - \{r\}$
  $F' := EXEC(r, F)$
  if$(F' \neq F)$
    $F := F'$
    $T := R$

Figure 8: Analysis of an AUTOSAR model.

6.5 Analysis of an AUTOSAR Model

The analysis of an AUTOSAR model is based on an iterative process that performs the abstract execution of all runnables in R, using an abstract context file. In the initial abstract context, the level of each port variable is fixed to the level of the port, and the level of every other variable is set to $\emptyset$. If during the analysis a level in the context file changes, all runnables must be re-executed.

The analysis uses an abstract interpreter, named EXEC, that analyses a single runnable. EXEC performs an abstract execution of the runnable starting from a context file F and producing a newcontext file F'. The state of a runnable is a tuple $\langle \sigma, q, F, M^\sharp \rangle$, where q is the code of the runnable, F is the abstract context file and $M^\sharp$ is the abstract memory and σ is the environment.

The analysis terminates when, starting from a context file, all runnables are executed and the context is not changed.

The main steps of the iterative analysis are shown in Figure 8, where F_0 is the initial context.

A the end of the analysis, the context file records the dependencies for ports of all the software components. The approach is conservative, in the sense that all possible dependencies for any real execution of the runnables are detected. False dependencies are possible, since in the abstract analysis all branches of control instructions are executed, even those that in real execution would have never been executed.

Example In the following, we report results of the application of the method to the example.

The analysis starts from Table 1, that reports the inital context file and local memories of runnables.

Let us consider the abstract execution of runnable 1. The `if` instruction causes the beginning of an implicit flow. According to the semantics rules, instructions in both branches of an `if` are executed under an environment set to the level of the condition of the `if`. On entering the `if` instruction, the environment is upgraded to the level of the variable (`RPort_DE1`). The implicit flow terminates at instruction (*), which is the first instruction out of the scope of the `if`. At

this point, the environment is downgraded to the level before the execution of the conditional instruction.

Table 2 reports the state of the memory at the end of the analysis of the three runnables.

Table 1: Initial context file and local memories.

Variables	Dependencies
Context file	
in1	{ in1 }
in2	{ in2 }
rtb.Add	∅
rtdWork.Delay_DSTATE_a	∅
rtdWork.Delay_DSTATE_m	∅
rtdWork.Delay_DSTATE	∅
rtdWork.Integrator_DSTATE	∅
irv1	∅
irv2	∅
irv3	∅
irv4	∅
out1	{ out1 }
out2	{ out2 }
out3	{ out3 }
out4	{ out4 }
Runnable R1	
Delay_n	∅
Runnable R2	
Delay	∅
SubtracterBuffer	∅
Runnable R3	
tmp	∅
*tmp	∅
OutputBuffer	∅

With our static analysis we can assert that the output port out4, irv2 and irv4 depend on both the input ports, meanwhile the irv1 only depends on input port in1. With a simple approach where all output variables of a runnable depends on all its input variables irv1 would depend on input port i2 too.

7 CONCLUSIONS

Security in automotive is becoming increasingly important and should be taken into account from the early stages of the system design.

In this paper we present a methodology for the verification of data secure flow in security annotated AUTOSAR models.

The method we present is based on information flow analysis and abstract interpretation. The analysis computes the lowest security level of data sent on a communication, according to the annotations in the model and the code of runnables. In particular, our approach for data flow analysis can be put at an intermediate level between a syntactic approach (Volpano et al., 1992; Bernardeschi et al., 2004) and a fully semantic one (Leino and Joshi., 1998). The approach is dynamic and thus allows us to be more permissive than a syntactic approach. Moreover, the approach is based on a transition system and thus has the advantage of being fully automatic.

Table 2: Final context file and local memories.

Variables	Dependencies
Context file	
in1	{ in1 }
in2	{ in2 }
rtb.Add	{ in1 }
rtdWork.Delay_DSTATE_a	∅
rtdWork.Delay_DSTATE_m	∅
rtdWork.Delay_DSTATE	∅
rtdWork.Integrator_DSTATE	{ in1, in2 }
irv1	{ in1 }
irv2	{ in1, in2 }
irv3	∅
irv4	{ in1, in2 }
out1	{ out1 }
out2	{ out2 }
out3	{ out3 }
out4	{ out4, in1, in2 }
Runnable R1	
Delay_n	∅
Runnable R2	
Delay	∅
SubtracterBuffer	{ in1, in2 }
Runnable R3	
tmp	{ in2 }
*tmp	{ in2 }
OutputBuffer	∅

On a security annotated AUTOSAR model, security properties can be specified and formally proved. Moreover, the possibility of automatically generate security components reduces errors in the development phase.

ACKNOWLEDGEMENTS

This work has been developed under the framework of the European project SAFURE (Safety And Security By Design For Interconnected Mixed-Critical Cyber-Physical Systems) under grant agreement No 644080. Moreover, the research has been supported in part by the PRA 2016 project entitled Analysis

of Sensory Data: from Traditional Sensors to Social Sensors, funded by the University of Pisa.

REFERENCES

Adelsbach, A., Huber, U., and Sadeghi, A.-R. (2006). Secure software delivery and installation in embedded systems. In *Embedded Security in Cars*, pages 27–49. Springer.

AUTOSAR (a). http://www.autosar.org.

AUTOSAR (b). https://www.autosar.org/fileadmin/files/releases/2-0/software-architecture/rte/standard/autosar_sws_rte.pdf.

Avvenuti, M., Bernardeschi, C., De Francesco, N., and Masci, P. (2012). Jcsi: A tool for checking secure information flow in java card applications. *Journal of Systems and Software*, 85(11):24792493.

Barbuti, R., Bernardeschi, C., and De Francesco, N. (2002). Abstract interpretation of operational semantics for secure information flow. *Inf. Process. Lett.*, 83(2):101–108.

Bernardeschi, C., De Francesco, N., Lettieri, G., and Martini, L. (2004). Checking secure information flow in java bytecode by code transformation and standard bytecode verification. *Software - Practice and Experience*, 34(13):1225–1255.

Bernardeschi, C., Del Vigna, G., Di Natale, M., Dini, G., and Varano, D. (2016). *Using AUTOSAR High-Level Specifications for the Synthesis of Security Components in Automotive Systems*, pages 101–117. Springer International Publishing, Cham.

Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskis, A., Roesner, F., Kohno, T., et al. (2011). Comprehensive experimental analyses of automotive attack surfaces. In *USENIX Security Symposium*. San Francisco.

Cousot, P. and Cousot., R. (1992). Abstract interpretation frameworks. *Journal of Logic and Computation*, 4(2):511–547.

D. E. Denning, P. J. D. (1977). Certification of programs for secure information flow. *Communications of the ACM*, 7(20):504–513.

Koscher, K., Czeskis, A., Roesner, F., Patel, S., Kohno, T., Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., et al. (2010). Experimental security analysis of a modern automobile. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 447–462. IEEE.

Leino, K. and Joshi., R. (1998). A semantic approach to secure information flow. In *Proc. 4th International Conference, Mathematics of Program Construction, LNCS 1422*, pages 254–271. Springer Verlag.

Lemke, K., Paar, C., and Wolf, M. (2006). *Embedded security in cars*. Springer.

MathWorks (a). Generate autosar-compliant code for multiple runnable entities (https://it.mathworks.com/help/ecoder/examples/autosar-code-generation-for-multiple-runnable-entities.html).

Nielson, F., Nielson, H. R., and Hankin, C. (2005). *Principles of Program Analysis*. Springer.

Sabelfeld, A. and Mayers, A. (2003). Language-based information-flow security. *IEEE journal on selected areas in communications*, 21(1).

Stephan, W., Richter, S., and Muller, M. (2006). Aspects of secure vehicle software flashing. In *Embedded Security in Cars*, pages 17–26. Springer.

Volpano, D., Smith, G., and Irvine, C. (1992). A sound type system for secure flow analysis. *Journal of Computer Security*, 4(3):167–187.

Wyglinski, A. M., Huang, X., Padir, T., Lai, L., Eisenbarth, T. R., and Venkatasubramanian, K. (2013). Security of autonomous systems employing embedded computing and sensors. *Micro, IEEE*, 33(1):80–86.