# Distributed IDSs for enhancing Security in Mobile Wireless Sensor Networks

Paola Inverardi, Leonardo Mostarda, Alfredo Navarra
Department of Computer Science
University of L'Aquila, Italy.
Email: {inverard,mostarda,navarra}@di.univaq.it

## Abstract

*We present an approach to provide Intrusion Detection Systems (IDS) facilities into Wireless Sensors Networks (WSN). WSNs are usually composed of a large number of low power sensors. They require a careful consumption of the available energy in order to prolong the lifetime of the network. From the security point of view, the overhead added to standard protocols must be as light as possible according to the required security level. Starting from the DESERT tool [2] which has been proposed for component-based software architectures, we derive a new framework that permits to dynamically enforce a set of properties of the sensors behavior. This is accomplished by an IDS specification that is automatically translated into few lines of code installed in the sensors. This realizes a distributed system that locally detects violation of the sensors interactions policies and is able to minimize the information sent among sensors in order to discover attacks over the network.*

## 1 Introduction

A Wireless Sensor Network (WSN) usually consists of a number of sensors of different modalities which, when combined with a microprocessor and a low-power radio transceiver, form a smart network-enabled node. The sensed data can be related to different applications but in terms of capabilities all the nodes that cooperate in the WSN can be assumed as homogeneous. Concerning the applications, medical services, battlefield operations, crisis response, disaster relief, environmental monitoring, premises surveillance, robotics are included. Because of the critical environments where such kind of networks may be used, we are interested in investigating security issues. Such kind of networks are in fact frequently subject to attacks by malicious intruders. Intrusions take place by a sequence of actions that aim to subvert the network system. There are many vulnerabilities and threats to a mobile WSN. They include outages due to equipment breakdown and power failures, non-deliberate damage from environmental factors, physical tampering, and information gathering (see [8] for an extended survey).

In this paper we are interested in the so called *active* kind of attacks where the detection of malicious behaviors can be locally detected by observing the traffic over the network (see for instance [7]). We focalize our attention on Intrusion Detection Systems (IDSs) that analyze the observable behaviors of the system.

More precisely we are interested in *Specification-based IDSs* [2, 3] which use some kind of formal specification to describe the correct behavior of the system. The detection of violations involves monitoring deviations from the formal specification. The advantage of this approach is the ability to detect previously unknown attacks at the expense of providing a formal specification of correct information flows.

Intrusion detection techniques are successfully applied to wired networks, however most of them are not suitable for the wireless context. The inadequacy of standard IDS in WSNs is a consequence of the open medium access, of the dynamic topology, of the cooperation needed among sensors (collaborative algorithms) and the lack of a fixed topology where all information flow. The detection of an attack may not be possible by only considering the traffic locally sent/received by a sensor. Therefore, there must be some form of correlation among the data received by the sensors. This would allow to discover attacks that are scattered over several sensors.

Besides the problems mentioned above, in mobile WSNs the detection has to address the problems imposed by the limited battery, restricted computational power of the sensors, low bandwidth and slower links. Moreover, mobility of the sensors can complicate the detection since the correct behavior of sensors is location dependent (i.e., the sensors correct behavior can change over the time). In summary, an IDS for a mobile WSN should: (i) be decentralized; (ii) minimize the traffic overhead; (iii) address the mobility problem.

Based on the DESERT tool [2] we verify the applicability and effectiveness of specification-based intrusion detection in a mobile WSN. We enhance the DESERT tool in order to build a peer-to-peer IDS for mobile WSNs. We describe the correct sensors communications by means of a formal specification. Such a specification is divided in different parts in order to distribute and share the workload hence reducing the energy consumption of single sensors. In order to validate and to explain in more details this approach we apply it on a recent proposed routing protocol, CoP [6], for mobile WSNs. Notice that, while the formal specification directly derives from the considered protocol, the method is completely independent of it, in the sense that it can be used on any kind of protocol for mobile WSNs in order to add security aspects.

## 2 DESERT Tool and mobile WSNs

The DESERT tool is a specification-based intrusion detection system that allows the monitoring of a distributed black-box component application (see Figure 1.(a) for a description of its basic steps). The application to be monitored is assumed to be composed by a set of components $\{C_1, \ldots, C_n\}$ that run concurrently and interact each other exchanging messages. Each component $C$ has associated a set of messages $\{m_1^C \ldots m_i^C\}$, with $i \geq 1$, (i.e., the component interface description) where $m_j^C$, with $1 \leq j \leq i$, encodes information about the type of communication, i.e., a request or a reply, the kind of service and its parameters and the (returned) data. In particular, $m_j^C = !a$ ($m_j^C = ?b$) denotes that the component $C$ sends (receives) a message $a$ ($b$) to (from) another component. The original DESERT tool assumes that: (i) each component of the system has a unique instance; (ii) a component cannot change location; (iii) a component cannot change its interface and behavior. From the point of view of mobile WSNs, from now on, we identify the black-box-components of the DESERT tool with the sensors. Being a black-box-component for a sensor means we do not have the code of the sensor but we can observe its interactions with the environment.[1] Clearly the previous assumptions cannot be now guaranteed. The more evident is (ii) since we are in a mobile environment while for (i), there are usually more sensors playing the same role. Concerning (iii), in general, in WSNs the sensors change their behavior according to their actual role in the network. The idea is usually to guarantee a balanced consumption of the energy of all the sensors participating in the protocol. As we are going to see in the next section, in the CoP protocol [6] the roles change depending on the actual positions of the sensors. We therefore remove assumptions (i), (ii) and (iii), hence obtaining a framework able to work with all those protocol for WSNs in which sensors have different behaviors according to a predetermined set of roles. The security aspects that we want to guarantee by this method concern the monitoring of the normal behavior of the sensors participating in a given protocol.

[1]Indeed we observe the interactions with its transmitting/receiving system, enabling it according to the specified polices.

## 3 Connectionless Probabilistic (CoP) routing

The CoP protocol [6] for routing on mobile WSNs assumes a uniformly distributed set of sensors inside a given area. The only thing that each sensor needs to know in order to participate in the CoP protocol is its own location and the location of the sink (the fixed infrastructure outside the sensed area). In such a model, a communication session begins when a sensor needs to inform the sink about some collected information of interest. The key idea of the CoP protocol is that the saving of energy is achieved not only by choosing an appropriate path between source and destination pairs but also by eliminating all the transmissions usually needed by other protocols to choose the next hop node or just to communicate the positions of the nodes (see for instance [4]). Clustering methods are used to reduce the number of needed hops to establish the required communication session and hence reduce the average routing time. To this end, a two-level communication model was proposed where each node is itself candidate to be either a normal sensor or a clusterhead.

Messages are routed on a virtual infrastructure represented by a grid covering the sensed area. Since the sensors are randomly spread on the area of interest, a distortion parameter called $ds$ is fixed to be the maximum distance from a virtual grid node ($VGN$) where the real sensor has to reside in order to candidate itself and become a clusterhead. Roughly speaking this means all sensors in the fixed range of a $VGN$ “believe” they are grid nodes. All the other remaining sensors are themselves associated to some $VGN$ just by the minimum distance, hence determining an area ($A_{VGN}$) associated to each $VGN$. Notice that if more than one sensor resides inside a described circular area, a standard local “leader election” is performed [5]. In [6] it is formally described how to estimate the right value for $ds$ in order to achive a high probability to have a sensor inside each $A_{VGN}$. The configuration can easily change with time, according to the degree of the sensors' mobility but each one can decide which is the closest clusterhead-area or if it is a clusterhead itself. If a sensor is a clusterhead, it can transmit the collected information to the next clusterhead-area in order to reach the sink. Since the transmission needed power of a non-clusterhead node is less than the one needed by a clusterhead, in order to prolong the lifetime of the entire network, a sort of rotation in the roles could be convenient. According to the frequency of the communications and the mobility of the nodes, if the network is characterized by high mobility, then every node frequently changes its status from clusterhead to non-clusterhead and vice-versa depending on its actual location and therefore mobility works in favor of a fair and uniform energy consumption in the CoP protocol.

## 3.1 Adapting the model

In the field of location-awareness and clustering protocols like CoP, we model a mobile WSN by a set of sensors $AH = \{C_1, C_2, \ldots, C_n\}$. Let $L \subseteq AH$ be a subset $\{l_1, l_2, \ldots, l_m\}$ of sensors identifying the clusterhead of a given protocol $P$. In other words, the sensors in $L$ characterize a set of areas $Ar = \{Ar_1, Ar_2, \ldots, Ar_m\}$ (clusters) where each area $Ar_i$ represents the portion of the sensed area where the corresponding $l_i$ plays the clusterhead role.

There can be various roles according to $P$, let $R = \{R_1, R_2, \ldots, R_n\}$ be the set of roles. Each $R_i$ has associated a set of possible messages $\{m_1^{R_i}, m_2^{R_i} \ldots m_k^{R_i}\}$ (i.e the role interface). In general a sensor $C_i$ can change its role from $R_i$ to $R_j$ as a consequence of: (i) changing the location; (ii) a message sent/received by the sensor.

**Definition 1** *Let C be a sensor instance.* $T_C = m_1^c\ m_2^c m_3^c \ldots m_k^c m_{k+1}^c \ldots$ *is a* local sensor trace of C *if each* $m_i^c$ *is a message that codifies either a request or a provided service of the sensor C.*

**Definition 2** *Given two local sensor traces* $T_{C_1}$*:* $m_1^{C_1} m_2^{C_1} m_3^{C_1}\ \ldots\ m_i^{C_1} m_{i+1}^{C_1}\ \ldots$ *and* $T_{C_2}$*:* $m_1^{C_2} m_2^{C_2}\ m_3^{C_2}\ \ldots\ m_k^{C_2} m_{k+1}^{C_2}\ \ldots$*,* a merge trace $T_{C_1} \oplus T_{C_2}$ *is a sequence defined by* $m_1 m_2 m_3 \ldots m_j m_{j+1} \ldots$ *where: (i)* $m_r \in T_{C_1} \oplus T_{C_2}$ *if and only if* $m_r \in T_{C_1}$ *or* $m_r \in T_{C_2}$ *(ii) for each* $m_i$ *and* $m_j \in T_{C_1} \cup T_{C_2}$ *if* $t(m_i) < t(m_j)$*then* $m_i$ *appears before* $m_j$ *in* $T_{C_1} \oplus T_{C_2}$.

According to the previous definitions, in a mobile WSN, we can recognize one architectural system trace for each defined area $A_{r_i}$. Each running sensor $C_i$ of the area $A_{r_i}$ has a role $R_j$ and defines a local component subtrace $T_{C_i}$. In particular, an architectural system trace is generated by messages exchanged among all the sensors contained in $A_{r_i}$. A global automaton models the *correct* exchange of messages among sensors playing different roles and residing in (i) the same area; (ii) an adjacent area. Our main purpose is to monitor the architectural system traces of each area in order to detect violation of the properties expressed by the global automaton.

## 3.2 DESERT Tool and CoP

Starting from the description of the CoP protocol we now point out some basic properties that should be guaranteed in order to obtain a fair behavior of the protocol. In what follows we distinguish the sending operation of a clusterhead

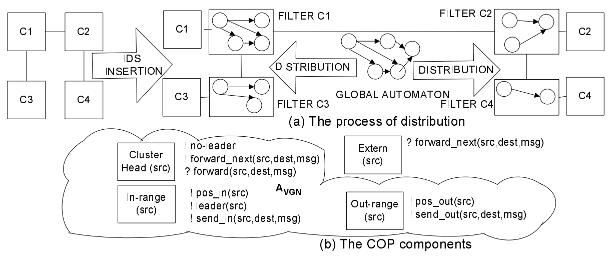

**Figure 1. Distribution process and CoP components**

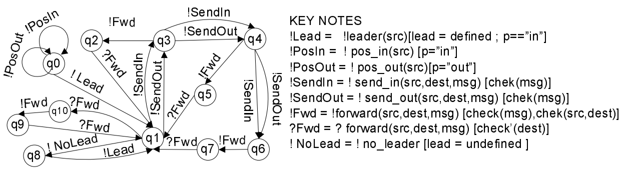
!PosOut
!PosIn
q0
!Fwd
q2
q3
!SendIn
!SendOut
q4
?Fwd
! Lead
!SendIn
!SendOut
!Fwd
q5
?Fwd
!SendIn
!SendOut
!Fwd
q10
?Fwd
q9
?Fwd
q1
! NoLead
q8
!Lead
?Fwd
q7
!Fwd
q6
KEY NOTES
!Lead = !leader(src)[lead = defined ; p=="in"]
!PosIn = ! pos_in(src) [p="in"]
!PosOut = ! pos_out(src)[p="out"]
!SendIn = ! send_in(src,dest,msg) [chek(msg)]
!SendOut = ! send_out(src,dest,msg) [chek(msg)]
!Fwd = !forward(src,dest,msg) [check(msg),chek(src,dest)]
?Fwd = ? forward(src,dest,msg) [check'(dest)]
! NoLead = ! no_leader [lead = undefined ]

**Figure 2. Global automaton**

from the one made by a normal sensor (non-clusterhead) just by referring to the first one as a *forward* operation.

1. For each grid node there must be at most one sensor playing as clusterhead.

2. When a finite amount of data has been collected by a clusterhead, it must be forwarded in the correct direction.

3. A clusterhead that changes its status to normal sensor due to a movement or because of the draining battery has to forward all the collected messages before its movement.

4. All messages forwarded by a clusterhead have to be received by the clusterhead of the adjacent VGN area.

5. When a clusterhead leaves its role a new sensor (if any in the area) has to take its role.

We formalize these properties by defining a state machine that will be given in input to our tool in order to produce the distributed "patch" for the sensors participating in the CoP protocol.

In Figure 1.(b) we show the four types of roles that each sensor can assume and the corresponding messages according to the described CoP protocol. Considering each $A_{VGN}$ we define the Out-range, the In-range and the Clusterhead sensors residing in it, and an Extern sensor representing the clusterhead associated to an adjacent $VGN$. The role Out-range Sensor(*src*) models a sensor located in a position *src* that is inside the $A_{VGN}$ but at distance greater than $ds$ from the corresponding $VGN$. The role In-range Sensor(*src*) models a sensor located in a position *src* inside an $A_{VGN}$ and at distance at most $ds$ from the corresponding $VGN$. The role Clusterhead(*src*) is the one played by the sensor that provides the transfer of the messages $msg$ towards the right sink. Finally, the role Extern models one of the clusterheads surrounding the current $A_{VGN}$. Figure 2 shows the global automaton related to the sensors based system of figure 1.(b). This automaton defines the correct sequences of messages inside an $A_{VGN}$. Our aim is to distribute the global automaton in a set of local automata that are assigned one for each sensor according to their actual roles. A local automaton constitutes the basis to build a filter that locally monitors the sensor instance interactions.

## 4 Distributing security features

In this section we describe the technique of filters generation roughly showed in Figure 1.(a) and we illustrate its applicability on the CoP protocol. The algorithm to distribute the global automaton creates a local one for each sensor role. The local automaton of the role $C$ constitutes the basis to realize a filter (we will denote it by $\Im_C$). In the following we describe the two main phases of the algorithm, i.e., local automata generation and dependencies generation.

### 4.1 Local automata generation

This phase considers in input the global automaton $A = (Q, q_0, I, \delta)$ and a $C$ sensor interface (i.e., the messages sent/received by $C$). The output is a preliminary version $\Im_C = (Q_C, q_{C0}, I_C, \delta_C)$ of the C-local automaton. $\Im_C$ is obtained by considering each rule $q_1 = \delta(q, m)$ defined in $A$. In the case that $m$ belongs to the interface of $C$ ($m \in C$) such rule is reflected in a $\Im_C$-rule $q_1 = \delta_C(q, m)$, the states $q$ and $q_1$ are added to $Q_C$ and the message $m$ is added to $I_C$. In other words, looking at the global automaton $A$, the interactions sequence that happens locally on a sensor $C$ are projected on the local automaton $\Im_C$. However the preliminary version of a local automaton is not sufficient to realize the correct monitoring. Our solution is to allow a filter to provide/accept context information by/to the other filters. We call such information exchanged among filters *dependency information*.

**Definition 3** *Let $\Im_C$ be the filter of the sensor role C. Dependency information is of the form $!f(m, \{D_1, D_2, \ldots, D_n\})$ or $?f(m', S)$ where $D_1, D_2, \ldots, D_n$ and $S$ range on the name of the sensor roles. The message $!f(m, \{D_1, D_2, \ldots, D_n\})$,* outgoing dependency*, is sent by $\Im_C$ to filters $\Im_{D_1}, \Im_{D_2}, \ldots, \Im_{D_n}$ in order to communicate that the C-sensor role message $m$ has been observed. The message $?f(m', S)$,* incoming dependency*, is an incoming information sent by filter $\Im_S$. With this information $\Im_S$ communicates to $\Im_C$ that it has observed a S-sensor role message $m'$.*

A dependency information allows filters to synchronize with each other to produce an architectural system trace that validates the property expressed by the global automaton. The dependencies are added to each local automaton by the dependencies generation phase.

### 4.2 Dependencies generation

The basic entities to place dependencies are synchronization and enabling states. A *synchronization state* of a global automaton defines a state that is exited by different transitions projected on different filters. In this case the filters have to synchronize so that exactly one of these transitions will be performed. By referring to the global automaton $A$ of Figure 2 the state $q_3$ is exited by the transitions $q_2 = \delta(q_3, !forward\_next(src, dest, msg))$, $q_4 = \delta(q_3, !send\_in(src, dest, msg))$ and $q_4 = \delta(q_3, !send\_out(src, dest, msg))$. The state $q_3$ is a synchronization state of $A$ since those transitions are projected onto the filters related to Clusterhead, In-Range and Out-range sensor, respectively. Dependencies are a means for filters to synchronize so that exactly one of them accepts its local sensor message.

An *enabling state* $q$ of $A$ defines a state $q$ that is entered and exited by transitions projected on different filters. The basic idea is that a filter, containing a $q$-exiting transition, should apply this rule only when a $q$-entering transition has been performed. This permits to correctly preserve the ordering imposed by the global automaton. In our example the state $q_1$ is an enabling one since it is entered by the transition $q_1 = \delta(q_0, !leader(src))$ and exited by $q_3 = \delta(q_1, !send\_out(src, dest, msg))$. This sequence of rules is projected onto the filters $\Im_{In-range}$ and $\Im_{Out-range}$, respectively. In this case the dependencies phase adds the rule $q_1 = \delta(q_0, ?f(leader(src), In))$ to the filter $\Im_{Out-range}$. Therefore $\Im_{Out-range}$ can move to the state $q_1$, by means of the rule $q_1 = \delta(q_0, ?f(leader(src), In))$. However this rule can be applied only when a filter $\Im_{In-range}$ has accepted the $!leader(src)$ message (i.e., the right ordering among the messages is imposed). We call such type of dependencies *enabling dependencies* since they are used (by a filter) to correctly enable the filters-local computations.

In the case of WSNs, due to the open access medium, some outgoing dependencies can be avoided. Let $?f(m, D)$ be the incoming dependency sent by the filer $\Im_D$ to the filter $\Im_C$. In the case that $\Im_D$ resides in the $\Im_C$-communication range, $\Im_C$ could observe directly the $\Im_D$-message $m$. Just when the filter $\Im_C$ resides "far" from $\Im_D$ the outgoing dependency $!f(m, C)$ must be explicitly added to the filter $\Im_D$. After the dependencies generation phase, the automaton describing a filter may still have disconnected parts. We use $\varepsilon$-moves [1] standard methods to link the disconnected parts in the right order obtaining a whole connected local automaton for each filter.

### 4.3 Mobility aspects

As already noticed a sensor can change its location as consequence of its mobility. This can change the role of the sensor in the network hence its filter should refer to a different automaton. For instance when a sensor moves from the Clusterhead role to the Out-range, its behavior must be checked over the corresponding automaton. It can happen that a state in the original automaton is not present in the new one. In our example, $q_9$ is not present in the automaton of the Out-range role. In order to solve this problem we propose two solutions. One is to define a mapping function that takes in input a state $q_x$ of a role $C'$ and a role $C''$, and outputs a state $q_w$ of the role $C''$ that corresponds to $q_x$ in $C'$. Such a function should be evaluated by the filters each time the corresponding sensor changes its behavior (role). The overhead is then given by the needed computations to evaluate such a function.

Another solution is to enrich the messages by making explicit the corresponding move on the local automaton in such a way that the receiver can understand which dependency on the local automaton is satisfied hence discovering its current state. Coming back to our example when the Clusterhead leaves its state $q_9$, before discovering which will be the current state over the automaton that describes the Out-range role, it will have to observe some other sensor sending the message $?f(!fn, extern)$ enriched with the information $q_9, q_1$. It will then be synchronized in the state $q_1$. The overhead in this case is given by the extra information we need inside each message. The right solution strongly depends by the given protocol. From the energy consumption point of view the first solution seems more suitable.

### 4.4 How to monitor malicious behaviors

The local automata are the basic specifications to build our distributed monitoring system. Each filter implements a local secure automaton and it is assumed to be interposed between the environment and the sensor it has to monitor. Before a filter can accept a local-sensor message has to (i) send the synchronization dependencies in order to acquire the right; (ii) if it gains the right then it has to send the enabling dependencies to correctly preserve the order imposed by the global automaton. Moreover, it sniffs its radio range and goes on when observing a message that matches with an incoming dependencies or a message performed by other filters of the same component type.

The filters monitor the local sensor behavior in order to detect when they violate the policy expressed by the global automaton. The overhead introduced by the filters distribution is due to the message exchanged among sensors residing on different $A_{VGN}$, since, in this case, the filters have to explicitly synchronize with each other. On the other hand this process permits also to discover eventual empty $A_{VGN}$ hence saving further messages with respect to the original CoP protocol.

Generally speaking, a sensor discovers an attack and inform its neighborhood when it observes a message that does not match the one it is expecting or a time out on the waited event is triggered. Notice that, if new sensors without the wrapping filters are introduced in the network, they are somehow forced to follow the right protocol since the "good" ones observe and reveal forbidden actions mismatching waited interactions.

## 5 Conclusions

In the context of IDSs, we have presented a new framework that outputs distributed secure protocols in the field of mobile WSNs. We have applied it to a recent proposed protocol for mobile WSNs, CoP [6]. We have shown how it is possible to wrap filters around the CoP sensors directly derived from the automaton given in input which describes the desired behavior of the system. The sensors are then synchronized, when needed, by means of messages exchange, hence they are able to detect intruders locally by themselves and by means of collaborations.

## References

[1] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley publishing company, 1979.

[2] P. Inverardi and L. Mostarda. A distributed intrusion detection approach for secure software architecture. In *EWSA 2005*.

[3] C. Ko, M. Ruschitza, and K. Levitt. Execution monitoring of security-critical programs in distribute system: A specification-based approach. *IEEE*, 1997.

[4] W.-H. Liao, J.-P. Sheu, and Y.-C. Tseng. GRID: A fully location-aware routing protocol for mobile ad hoc networks. *Telecommunication Systems*, 18(1-3), 2001.

[5] N. Malpani, J. Welch, and N. Vaidya. Leader election algorithms for mobile ad hoc networks. In *DIALM 2000*.

[6] J. A. McCann, A. Navarra, and A. A. Papadopoulos. Connectionless Probabilistic (CoP) routing: an efficient protocol for Mobile Wireless Ad-Hoc Sensor Networks. In *IPCCC 2005*.

[7] J.-M. Orset, B. Alcalde, and A. Cavalli. An EFSM-based intrusion detection system for ad hoc networks. In *ATVA 2005*.

[8] A. Perrig, J. Stankovic, and D. Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.