TinyWIDS: a WPM-based Intrusion Detection System for TinyOS2.x/802.15.4 Wireless Sensor Networks

Luigi Pomante
DEWS, University of L'Aquila
luigi.pomante@univaq.it

Walter Tiberti
DISIM, University of L'Aquila
walter.tiberti@graduate.univaq.it

Fortunato Santucci
DEWS, University of L'Aquila
fortunato.santucci@univaq.it

Marco Pugliese
DEWS, University of L'Aquila
marco.pugliese@guest.univaq.it

Lorenzo Di Giuseppe
University of L'Aquila
lorenzo.digiuseppe88@gmail.com

Marco Santic
DEWS, University of L'Aquila
marco.santic@univaq.it

Luciano Bozzi
RhoTecnology
Via dei Mille, 41A, 00185 – Roma
luciano.bozzi@rotechnology.it

ABSTRACT
Last years have seen the growth of interest for Middleware (MW) exploitation in distributed resource-constrained systems like Wireless Sensor Networks (WSN). Available MW platforms usually provide an Application Layer (AL) with different basic services but no security services. In such a context, this paper describes an existing TinyOS2.x-based MW tailored to IEEE 802.15.4 WSN (Agilla2) and an existing Intrusion Detection System (IDS) based on a Weak Process Model approach (WIDS). Then, the paper reports the main issues related to the implementation of WIDS on TinyOS2.x/802.15.4 (TinyWIDS) also providing some experimental results. Finally, the paper proposes a strategy for the integration of TinyWIDS into Agilla2.

KEYWORDS
WSN, IDS, Weak Process Model, Mobile Agent Middleware

1 INTRODUCTION
Wireless Sensor Networks (WSN) are commonly exploited in monitoring applications. Such kinds of networks are composed of *sensor nodes* with severe HW limitations [1]. Typical WSN applications require sensor nodes distributed in space with at least one node, called *sink node*, connected to a *base station*, which collects data from the others. The final application greatly influences the design of the network architecture. For such reasons, the last years have witnessed the growth of methods to avoid to application developers the management of network parameters. One of them is to use middleware-based approaches for application development [10]. In fact, a *Middleware* (MW) reduces the development cost of WSN applications by providing a intermediate software layer that carry out low-level network and resource management tasks, leaving to the developers only the task of providing the application-dependent code. This paper focuses on a special kind of WSN-MW, the *(mobile) agent-based MW* (MAMW), which exploits small pieces of code (*agents*) that can move inside the network. This peculiarity gives the possibility to reprogram the nodes once deployed, or to relocate the code, without losing continuity of services.
A debate living in WSN world is related to the security of the WSN platform. Given the limited HW resources, applying security techniques poses multiple challenges. This issue is often solved by providing *no security* at all or providing a lightweight solution. This paper assumes that the application cannot avoid the use of security techniques. So, this paper focuses on one specific aspect of security, the development of an *Intrusion Detection System* (IDS). Also, to provide an integrated yet flexible solution, the paper exploits a famous MAMW for WSN, running on the *TinyOS2.x* [4] operating system and exploiting the *IEEE 802.15.4* protocol [5].
The structure of the paper is the following: Section II describes background and motivations, Section III presents the exploited MAMW, and Section IV introduces the adopted IDS solution. Then, Section V describes IDS implementation while Section VI outline its integration of into the considered MAMW. Finally, Section VII completes the paper with some conclusions and future works.

2 BACKGROUND

2.1 Security in WSN
Granting security in a WSN is harder than in traditional wireless networks since WSN suffers of all the same problems but available HW resources are severely

constrained. To solve this problem, usually, ad-hoc techniques are used (e.g. [6]).

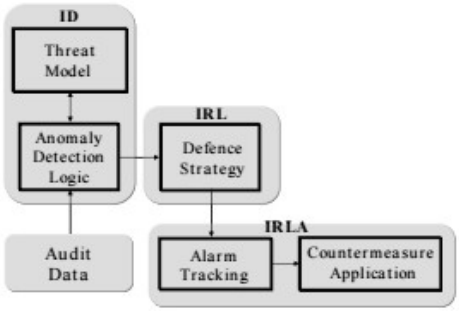

Figure 1. IDS Architecture

2.2 Intrusion Detection Systems

Ensuring availability in WSN is a central problem. Typically, availability is asserted by auditing network activities, detecting potential threats and reacting opportunely through an *Intrusion Detection System* (IDS). An IDS denotes a system capable of detecting intrusions and attacks in the form of malicious control and data messages [7]. An IDS is typically formed by three components: *Intrusion Detection* (ID), that deals with the detection of network intrusions, the *Intrusion Reaction Logic* (IRL), that schedules the priorities for actions on compromised nodes according to a specific defensive strategy, and the *Intrusion Reaction Application*, which applies the appropriate countermeasures. Figure 1 shows the general IDS architecture where three macro components can be identified:

- the *Identification* (ID) component containing the *Threat Model* (TM) and the *Anomaly Detection Logic* (ADL);
- the *Intrusion Reaction Logic* (IRL) component where the *Defence Strategy* (DS) is implemented following the directives of the IRLA;
- the *Intrusion Reaction Logic Application* (IRLA) component which contains the definition of the countermeasures and the *Alarm Tracking*.

An IDS can be classified into three categories: *anomaly based*, *misuse based* and *specification-based* [8]. *Anomaly-based* IDSs are based on the assumption that an attacker will show an *abnormal behavior* that is then detectable by the IDS. *Misuse-based* IDSs have an up-to-date *database of signatures* of known attacks. Finally, *specification-based* IDSs have instead a set of rules that are used to decide if there is an intrusion.

3 AGILLA2

Agilla MW [2] is a *mobile-agent middleware* for WSN originally written for TinyOS 1.x. An *Agilla* application consists of multiple *agents,* small pieces of code capable of running and migrating among sensor nodes. Agilla architecture is shown in Figure 2. The data model used in Agilla is the *tuplespace*, a set of *tuples* shared among agents. A tuple is a *key-value* pair, whose data is strongly typed. Also, Agilla has a *neighbor list*, a table containing the current neighbor node addresses. Agilla defines its own *Instruction Set Architecture* (ISA) which allows agents to be written in a stack-based assembly-like language.

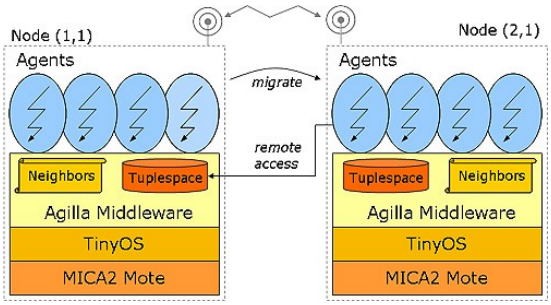

Figure 2. Agilla architecture

A last important concept exploited by Agilla is the *reactions*. An agent can define one or more pieces of code that are executed only when a certain condition is met, which is useful for implementing *asynchronous* operations. Since this paper focuses to the newer and improved *TinyOS 2.x*, it considers a proper porting of *Agilla* (called *Agilla2)* [3].

4 WIDS

The *WSN Intrusion Detection System (WIDS)* is a *Weak Process Model*-based (WPM-*based*) IDS [8] designed to work in WSN environments. The WPM is a *Hidden Markov Process* where the many-to-many relationship between *observables* and *states* (which represent the possible phases during an attack) is made deterministic by assigning to state transitions *0* or *1* weights rather than using probability. This allows to simplify the cost of evaluating the presence of threat making so possible to implement a *misuse-based* IDS in a WSN.

4.1 Threat analysis

Before describing with more details WIDS, it is needed to identify a set of possible threats in WSN to be considered for the detection. The main goal is to estimate the physical properties (i.e. *monitorables*) that allow to detect an anomaly and to produce events (i.e. *observables*) to be taken into account. In particular, WIDS focuses on main threats listed on Table 1.

Table 1. Common attacks

PHY layer	MAC Layer	Network Layer
Constant Jamming	*Frame spoofing*	*Sinkhole*
Receptive	*Frame*	*Wormhole*

Jamming	*tampering*	
Reactive Jamming	*Backoff-manip.*	*Sybil attack*
Random Jamming	*ACK flooding*	*Generic MitM*

4.2 Threat Model and Detection Logic

WIDS uses a *Finite State Machine* (FSM) approach to describe the WPM. The FSM is modeled through a weighted and oriented graph structure, in which the nodes represent the different states while the edges represent the possible state transitions. During the execution of the monitored applications, WIDS listens to events, evaluating (through an *attack database*) possible state transitions on the WPM. When a dangerous state is reached, the sequence of events is notified to the IRL.

Detection Logic. During the execution of monitored application, WIDS keeps track of the possible states in which the system could stay. As observables are caught, WIDS updates the list of states, discarding every state not conform to the observables sequence. This process allows WIDS to compute multiple estimated state sequences (*state traces*) dynamically, which are compared and evaluated with the state sequences of known attacks.

Threat Score. The actual intensity of the attack is evaluated by introducing a *threat score*. In [8] two classes of attack have been introduced: *LPA* (*Low Potential Attack*) and *HPA* (*High Potential Attack*). While threats are confined in LPA states, no immediate reactions should be issued. Instead, when in a HPA state, a reaction should be performed on the score associated to the estimated state trace. The total threat score is computed by summing the weights assigned to the states in all possible state traces at a particular time. Given the temporal sequence of observables, Figure 3 shows the state estimation timeline. The nodes contain the *state identification number* and the list of producible observables in that state. On the edges, the threat score associated to the state transition is shown.

Following the example, after the *reset state* (green), the possible state traces are those ending with state 1 and 4. When the first observable (1) is notified, state 4 is discarded since it cannot produce such an observable. At t_1 the observable 3 makes WIDS add two new state traces, ending with state 2 and state 3, while at t_2, as a consequence of the observable 4, the state traces ending with state 5 and state 4 are added. The process continues as described, but, since state 5 is an HPA, the IRL is notified with the threat score computed.

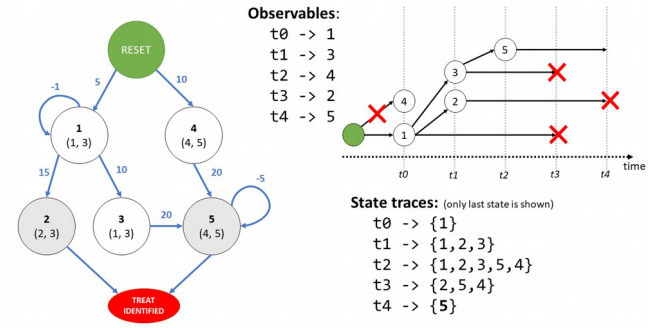

Figure 3. Example of WPM and state estimation.

5 TinyWIDS: Implementation & Validation

This section describes *TinyWIDS*, the implementation of WIDS on a TinyOS2.x WSN platform by showing a UML-conformant model, the TinyOS components breakdown, and some validation results.

5.1 Implementation

TinyWIDS implementation is built upon the *TKN154* MAC layer implementation [11] provided inside the TinyOS 2.1.2 package. Figure 4 shows the high-level UML model of TinyWIDS.

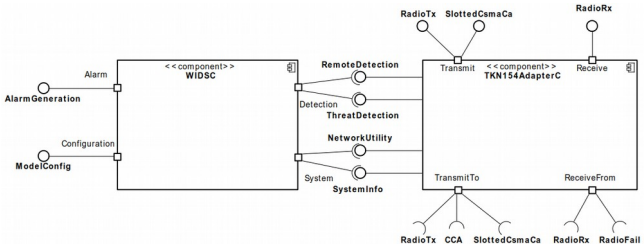

Figure 4. TinyWIDS model (UML)

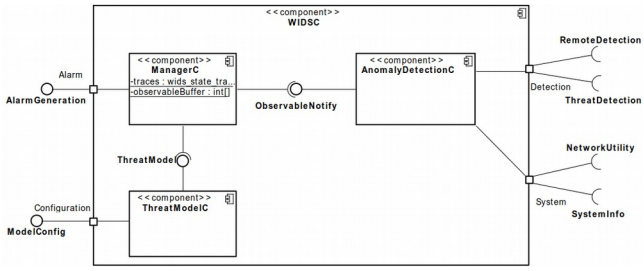

Figure 5. Internal architecture of WIDSC

Components. The anomaly rules detection step has been decomposed in two phases, resulting in two components. An application-specific component is responsible to translate network events in a set of predefined events that a second, more general, component can use to evaluate anomaly rules. In addition, *Anomaly Detection Logic* (ADL) and the *Threat Model* (TM) have been implemented as distinct

components, so that it is possible to reconfigure TM at compile-time through a configuration file describing the WPM. The architecture of resulting IDS is presented in Figure 4. The first component (*TKN154AdapterC*) lies between the MAC and physical layer. It redirects MAC requests to the radio driver and the events produced by radio driver to the MAC, signaling those events to the IDS. Then, as shown in Figure 5, WIDSC is composed of three sub-components. The *AnomalyDetectionC* component receives messages containing the information needed to identify the anomalies, transmitting new observables to a controller component. The controller (*ManagerC*) implements the core IDS functionality, such as the state evaluation algorithm. It strictly cooperates with the TM, represented by *ThreatModelC*.

Data structures. In order to represent the WPM structure (which is modeled by a graph), it has been used the *adjacency lists*. This representation guarantees good performances, since a neighbor of a node can be found with a asymptotic notation cost of $O(\delta(n_i))$ where $\delta(n_i)$ is the outgoing degree of a node n_i. The drawback of such a representation is the high memory occupation in the worst-case scenario (i.e. a *fully-connected* graph). So, since TinyWIDS can dynamically create multiple state traces, a *double-linked list* data structure has been used to store them in order to gain better performances for insertion and deletion of traces. In addition, in order to lighten the memory impact of state traces, it has been stored only the last state of each state trace.

5.2 Validation

In order to validate the effectiveness and the performance of TinyWIDS, two aspects have been considered while running it on *IRIS* sensor nodes [9]: the capacity to detect anomalies and the impact of the implementation on the protocol stack.

Test scenarios. Two of the most representative test scenarios are listed below:

- A *jammer* realized by means of an *IRIS* sensor node programmed to continuously transmit frames at the maximum power with a random payload and ignoring all the mechanisms relative to the *Clear Channel Assessment* (CCA).
- A *replay* attacker realized with a device listening to the channel and retransmitting the received frames with an increased sequence number.

Effectiveness. Tests results have shown that WIDS is able to detect both these two attacks. In particular, the *jamming attack* is identified correctly as a *constant jammer* most of the times, while in some tests, it is identified as *deceptive*, *reactive*, or *random jamming* attack. This is an expected result since frames are randomly sent and collisions happen randomly due to the fact that the jammer transmissions do not conform to the CSMA/CA protocol used in the IEEE 802.15.4. This ambiguity is instead not present in the *replay* attack scenario, which is correctly detected in all the tests.

Performance. In order to measure the performance overhead introduced by TinyWIDS, it has been considered that, upon the reception of a frame preamble, the radio chip usually generates an interrupt, which is handled by an interrupt service routine (ISR) that firstly captures a timestamp for network synchronization using a 62.5 kHz timer. In transmission, this timestamp is captured as soon as the preamble of the frame has been sent. In both cases the timestamps offer a good indication of when MAC receives or sends, respectively, a frame. Tests have been made both in presence and in absence of attacks with a couple of devices on a series of 50 detections during the process of association/disassociation to the network with beacon order 8 and superframe order 7 [5]. Table 2 reports average values related to the measurements.

Table 2. Performance Analysis

	Rx	**Tx w/o CCA**	**Tx Slotted**
No WIDS	2.288 ms	0.896 ms	3.2 ms
w/ attacks	2.544 ms	0.896 ms	2.8 ms
w/o attacks	2.896 ms	0.976 ms	3.616 ms

6 INTEGRATION IN AGILLA2

TinyWIDS implementation described in the previous section works very well when running as a stand-alone application installed on one or more WSN nodes (*ID* nodes) or when integrated into an applicative WSN SW. However, these approaches can limit the flexibility and the scalability of the WSN because the ID nodes must be chosen in the design phase and it can be difficult to later modify their behavior, position or number. So, in order to tackle this problem, this paper presents also an integration strategy, as an enhancement of the one proposed in [8]. The goal is to exploit *Agilla2* to distribute WIDS in a WSN in form of *agents*. This allows to have, virtually, the IDS feature in any node which requires it, only when needed, with the added benefit of a stronger *cooperation* among the agents and the nodes inside the WSN. In Figure 6, a brief schematic of the proposed integration is shown. Here, the WSN is programmed with a modified version of Agilla2 that incorporate all the static components of WIDS (i.e. WIDSC), while agents implements the dynamic components of WIDS which can travel from node to node seamlessly. The communication between Agilla2 and WIDSC is managed by a component called *IntrusionDetectionC*. Another component, called *IntrusionDetectionSystemC*, is used to retrieve the total score and the type of the attack, creating one or

more tuple with the data needed by an agent to wake up and react to the reported events. Cooperation is achieved by using an agent to write the attack information tuple in a nearby node tuplespace; upon the insertion of the tuple, *IntrusionDetectionC* reads the information and signals it to WIDSC so that it could update the state of the system. This integration permits a fine grain identification of anomalies in the network and the possibility to react in front of an intrusion simply injecting new logic in the network from the PAN coordinator. A possible scenario based on Figure 6 is the following one: a sensor node detects an anomaly and signals the alarm *Al[k]* (*k* represents the step of detection) to the ARL (*Alarm Reaction Logic*) agent. Then, the ARL agent sends the information to the PAN coordinator, which uses it as input for local IDS TM and signals an alarm to its ARL agent if it reveals a threat for the network. The logic that runs on the coordinator ARL agent activates the IRL (*Intrusion Reaction Logic*) agent. At this point, IRL agent migrates on the device that has detected the anomaly to counter the attack. At the end, the IRL agent is removed.

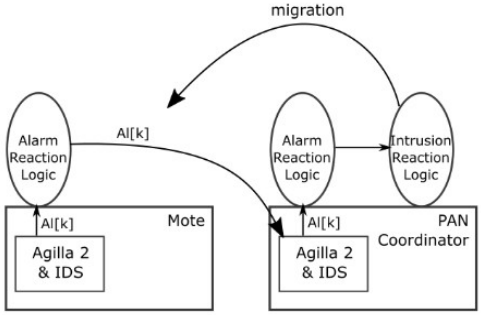

Figure 6. "WIDS on Agilla2 MW" Scenario

6.1 Current Integration Issues

Agilla2, as shown in [3], has a relatively high program memory occupation for common WSN node architectures. In the tests performed on common nodes architectures (i.e. *Telosb*, *MicaZ* and *IRIS* [9]), most of the selected platforms, had no enough available program memory to store concurrently TinyOS, TKN154, Agilla2 and TinyWIDS. So, current activities are focused to strip down some less useful components from Agilla2. Actual results are quite satisfactory since the memory occupation is around 120KB of program memory and 7KB of RAM while IRIS nodes offer 128 KB of program memory and 8 KB of RAM. However, there are still some stability issues that have to be solved before to be able to present meaningful validation results in such a direction.

7 CONCLUSIONS

This paper has presented TinyWIDS, a TinyOS2.x based implementation of a *misuse-based Weak Process Model* IDS (*WIDS*) ideal to be exploited into WSN platforms. The general features of an IDS, the basic technologies, and the architecture of TinyWIDS have been described focusing also on the validation scenarios, the obtained results, and the performance of the IDS implementation. Finally, an integration strategy towards an agent-based MW has been outlined. Current integration issues have been briefly described. They represent also the next steps to be faced in the near future.

REFERENCES

[1] Mark Hempstead, Michael J. Lyons, David Brooks, and Gu-Yeon Wei. "Survey of Hardware Systems for Wireless Sensor Networks", Journal of Low Power Electronics Vol .4, 1–10, 2008

[2] Fok, C.-L., Roman, G.-C., Lu, C., "Agilla: A Mobile Agent Middleware for Sensor Networks". Accepted to ACM Transactions on Autonomous and Adaptive Systems Special Issue.

[3] L. Pomante, L. Corradetti, D. Gregori, S. Marchesani, M. Santic, W. Tiberti. A Renovated Mobile Agents Middleware for WSN - Porting of Agilla to the TinyOS 2.x Platform. IEEE RTSI 2016.

[4] Levis, Philip, Sam Madden, Joseph Polastre, Robert Szewczyk, Kamin Whitehouse, Alec Woo, David Gay, et al. 2004. "TinyOS: An Operating System for Wireless Sensor Networks". Ambient Intelligence. Springer-Verlag.

[5] IEEE Standard for Information Technology, Telecommunications and Information Exchange Between Systems- Local and Metropolitan Area Networks-Specific Requirements Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs). Rapp. tecn. 2006, 0 1–305.

[6] L. Pomante, M. Pugliese, S. Marchesani, F. Santucci. "Definition and Development of a Topology-based Cryptographic Scheme for Wireless Sensor Networks". Sensor Systems and Software - Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2013.

[7] D.E. Denning, "An Intrusion-Detection Model", IEEE Transactions on Software Engineering, vol. SE-13, no. 2, February 1987.

[8] L. Pomante, M. Pugliese, S. Marchesani and F. Santucci, "WINSOME: A middleware platform for the provision of secure monitoring services over Wireless Sensor Networks," 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), Sardinia, 2013, pp. 706-711.

[9] http://www.aceinna.com (2017)

[10] Mohammad Abdur Razzaque, Marija Milojevic-Jevric, Andrei Palade, and Siobhán Clarke "Middleware for Internet of Things: A Survey", IEEE Internet of Things Journal, Vol. 3, No. 1, Feb 2016

[11] Jan-Hinrich Hauer "TKN15.4: An IEEE 802.15.4 MAC Implmentation for TinyOS 2", TKN Technical Report TKN-08-003, Editor: Prof. Dr.-Ing. Adam Wolisz, Berlin, March 2009