Optimized Resource Access Control in Shared Sensor Networks

Christophe Huygens, Nelson Matthys, and Wouter Joosen

IBBT-DistriNet, Department of Computer Science, K.U. Leuven
Celestijnenlaan 200A B-3001 Heverlee, Belgium
{christophe.huygens,nelson.matthys,wouter.joosen}
@cs.kuleuven.be

Abstract. The security concern in wireless sensor networks is driven by the need for increased assurance regarding the system. In this light, research on protecting the network from threats originating from the hostile outside has been ongoing. Additionally, many real world applications of sensor networks move away from the monolithic application model – node capabilities need to be shared among different applications of different actors. This view introduces additional security requirements. This paper addresses controlled usage of resources, a primary security requirement in case of sensor sharing. A distributed reference monitor is proposed as the enforcement mechanism. The monitor is policy-driven which enables lightweight run-time control of the resource accesses. Resource constraints as well as current programming and operational models are respected through use of a selective injection strategy based on code rewriting during pre-deployment. Code rewriting is controlled by aspect-oriented constructs. The approach is validated by a research prototype.

Keywords: Sensor Network, security, monitor, policy, aspect-oriented.

1 Introduction

As the field of wireless sensor networks (WSNs) matures and technologies are deployed in multiple real-world business scenarios, additional challenges are uncovered that have not been significantly addressed at this time. In many scenarios WSNs are moving away from the mono-application, data-gathering use cases that have been archetypical for the space and have been driving the research problems. The role of the WSN devices in more complex business scenarios is not merely data-centric but expands to the execution of a localized part of the holistic application. This localized part is characterized by more complex functions such as local decision support or behavioral change in response to a changing node environment as dictated by the application logic of a business actor. As such the sensor network infrastructure is becoming another tier of enterprise information technology by providing a general-purpose but limited execution capability ideally suited for a wide range of scenarios exploiting the advantages of WSNs. Integration in the enterprise infrastructure and the specific nature of WSN however require addressing long-standing challenges such as management and security in new ways.

Within enterprise WSN scenarios many applications, launched by different business process partners deserving various degrees of trust, compete for essentially the same resources. We contribute in this paper by providing a strong case for additional security solutions complementing the low-level, network-layer security mechanisms currently available for outsider protection. We identify the resources on the node or within the middleware as in immediate need of additional protection in real-world scenarios and propose a security subsystem to protect the resources, an operational model on how to organize this security and an approach for seamless integration in the application life cycle. We propose and detail the key elements of our our security solution: the distributed policy engine and the injection mechanism that correspond to the identified requirements and validate through a research prototype. The paper is structured as follows. Section 2 provides motivation and the requirements for additional resource protection are identified and detailed in section 3. The solution, consisting of both the security mechanism and the model for deployment and operations is presented in section 4 emphasizing the aspects specific to WSNs. In section 5, feasibility is demonstrated by documenting implementation, results and tool chain of the prototype. Wrap-up consists of related research, future work and conclusion.

2 WSN Security Evolution

Many WSN applications have concentrated on gathering data, for example in a setting of wildlife monitoring [1]. Over time, these applications have become prototypes for typical sensor deployment where data is collected, processed by filtering and/or aggregation and conveyed to backend systems where the primary business logic is concentrated. Even though multiple sensing applications can be operating at the same time, the constituting infrastructural components (nodes, gateways and backend) are operating as a unit under control of the application developer who is implicitly responsible to resolve conflicts between parts of the application.

In today's business setting more flexible usages schemes are of key importance, as different actors own parts of the infrastructure shared by multiple applications. Even if the marginal cost of a sensor would drop to zero, it is impractical and unrealistic to expect each application stakeholder to deploy a WSN. As such, sharing is merely an extension of the long-standing evolution exemplified by sensor middleware decoupling infrastructure and application pool, such as done in MiLan [2]. Sharing happens in a similarly decoupled system, but recognizes that real-world systems, next to being subject to global optimization, consist of applications that can only be partially trusted by the WSN infrastructure.

Consider the setting of a logistics scenario. A powerful node can be attached to each container. Given cost and complexity of deployment and management, container ownership and dynamics of the business process it is mandatory that many actors of the process will interact with this node and consume resources. Several parties will want supply chain visibility, dictated by legal regulations or by the business process. For fast track processing the container owner needs to prove manifest validity and container integrity - including the protection of all associated information systems (dictated by C-TPAT, the Customs-Trade Partnership Against Terrorism or the European Authorised Economic Operator certificate). In the port, the unloading company actor requests stationary node support for localization of individual parcels within the 40 foot container space. A pharmaceutical end-user counts on the container node for critical environmental control of perishable goods and cold chain visibility (USP General Chapter <1079>). These functions are pushed into the WSN infrastructure driven by the need for fresh and trustworthy sensing and actions. Many tasks need to be handled by the WSN on behalf of the various actors and the applications and the WSN infrastructure are increasingly decoupled. The node on the container is a generic execution platform for sub-applications of the logistics actors that change dynamically. The applications reflect interests of different business actors and strong demand is present for sharing of the resources of the container node among those applications. Yet, given the tight nature of available resources, a control component must be present that governs and actively controls this resource sharing to provide a sufficient level of assurance to the parties involved – typically a system security function [3].

3 Control Requirements

The resource control solution that enables sharing in WSNs must be implemented in a secure and transparent way. The technical resulting requirements are well-known. However, the WSN target platform dictates additional requirements less present in classical systems.

The *traditional (formal) qualities* of resource control hold [4]: (i) complete mediation: does the component effectively and provably execute all intended controls. (ii) isolation: is the control component, being a critical security function, protected from the rest of the system. (iii) verification: provably correct behavior of the resource control function and its policy enforcement actions.

The memory and CPU resources within the WSN being equally scarce, fine-grained control over deployment of the resource control component is needed. For example an administrative owner of a node may not want to restrict sharing of resources on his nodes, or on the other hand can decide on exclusive use of a node for his own trusted applications. Conversely, some applications may not need to access protected resources and therefore do not need to be policed. If there is no need for resource controls its inclusion in the system should be avoided. So the control solution must offer *selective deployment* as to minimize resource consumption.

The functional and security requirements of an application have different change cycles. The shared WSN infrastructure needs to respond to the aggregate of all the changes requested by all applications. This is reinforced by the dynamic nature of the WSN environment that will cause the sharing goals of the infrastructure to change frequently, for example based on available resources. Thus, *support for small and incremental run-time changes to the sharing control policies* is required.

Finally, the application developer is badly placed to judge and code on sharing since only the administrative owner of the WSN, the actor that finally decides on resource allocation, has the overview and can set the scene. Additionally, segregation of duties calls for factoring the security components out of the application. This is not unique to WSNs. The developer, rarely well versed in security, is ideally not aware of his application being subject to security enforcement. To enable continued use of the existing application code base, we should be able to introduce the control component within the existing application-platform combination with as little change as possible to the application code. This raises the requirement of making our solution *non-intrusive* with respect to the development cycle and code.

4 Description of the Control Solution

To describe the proposed solution, we first sketch the system model context. Next we present the key elements: the reference monitor instantiated by a policy-driven enforcement engine, and selective and non-intrusive pre-deployment injection as the life cycle integration strategy. The policy-driven enforcement engine addresses the requirements of secure control and incremental change, whilst pre-deployment injection deals with selective and non-intrusive inclusion.

We then elaborate how the control subsystem can be architecturally and operationally integrated within the WSN environment. The reader is referred to literature for other elements of the solution such as formal aspects of the requirements [5], as well as already existing low level security functions needed to complete the overall implementation of the solution [6].

4.1 System Model Context

A WSN architecture consists of multiple layers and tiers [7]. From a development perspective, layers of increasing functionality are available to the application ranging from the hardware over run-time support and services in middleware to the programming abstraction. Horizontally, WSNs consist of devices of increasing capabilities ranging from the smallest sensor platforms over gateways to the back-end systems. Controls can be placed anywhere in these two dimensions where resources are available and control is desired. For the purpose of placement of the security functions we consider the WSN a white box – all components are explicitly visible and accessible – in contrast to some programming models that aim to hide some of the underlying infrastructure complexity [8]. Such hiding may hinder optimal security deployment by shielding lower level functions, such as resource access, unless complemented by some level of detail in deployment descriptors.

The system model used is simple (Figure 1). It consists of backend systems connected to the WSN by the gateway. The sole purpose of the gateway is to provide connectivity between a backend and a WSN node. Within the system, we identify 2 important actors: the *application owner* and the *administrative owner*, reflecting the decoupling from the application and resource pools. Applications consists of application components present on nodes and the backend of the application actor. The administrative owner actor is responsible for all management activity in the WSN such as deploying application components on the nodes.

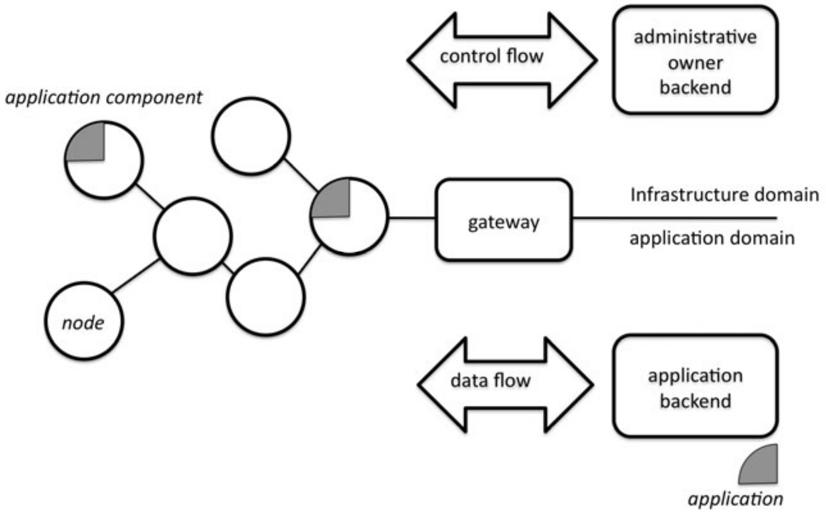


Fig. 1. System model

4.2 The Policy Engine and Reference Monitor

The security element that matches an object with an access operation is typically referred to as a reference monitor [4]. When an application has rights to a resource, the reference monitor allows usage as per policy. The policy is a time and context dependent description of the decision tree. Note that this not a universal monitor of code behavior such as based on automata [5], since the focus of this research is resource control. The presented solution is a distributed reference monitor system, with selective inclusion of a local enforcement component on the WSN node as dictated by backend policy analysis at pre-deployment We refer to the local component as the "policy engine". We also stress the difference between the "security monitor" (equally referred to as "reference monitor") as mechanism and "monitoring" as a post-factum security strategy.

4.3 Trusted Computing Base and Threats

The TCB (trusted computing base) of the WSN, the ensemble of functions that provide the reference monitor, must be protected against attack. Many taxonomies of threat models have been presented [6][9]. These threat models devised for both "outsider" and "insider" attacks remain valid, but now need to be applied in the context of the application of a reference monitor. We can assume the resource-rich gateway/backend environment to be trustworthy since traditional TCB solutions such as those based on resource-intensive PKI schemes can be applied. But for node-level resource protection, since the TCB extends over the WSN as a holistic unit, WSN network level security becomes critical. For example, the administrative owner of a WSN must distribute the access control matrix for the reference monitor securely. Providing for reliable, secure authenticated policy updates in the resource-constrained

WSN environment is challenging but available [10] – the TCB itself is incremental to these existing security mechanisms and leverages the state-of-the-art thereof.

4.4 Integration by Pre-deployment Injection

The dominant view of the resource reference monitor is that a reactive element matching application credentials with objects at run-time, as part of the operating system or middleware. But in a generalized view, other implementation strategies can be valid as well - a security monitor can analyze for illegal operations before actual run-time, for example at compile time, or pre-deployment by byte code verification. When devising solutions for WSNs, the dynamic and resource-limited nature is of primary concern. To address the resource scarcity we have to make maximum use of the resource rich backend systems, and offload as much as possible of the security function to the pre-deployment (code analysis and processing) and/or post-runtime phases (monitoring). The approach is preferably non-intrusive with respect to the current development cycle. Therefore a solution based on selective injection of interception code is proposed, using application code analysis before deployment, as opposed to standard library-style integration of the monitor into the middleware platform, the latter often creating unnecessary overhead. This means that the monitor will operate at run-time, but only on the nodes where it has previously been injected at pre-deployment time. If during the lifetime of the system, an additional monitor is needed where it was previously not yet available such as on an unshared node, an extra provisioning step will be needed.

4.5 Operational Model

Application deployment follows these stages: (1) upon development the application owner submits its code to the administrative (infrastructure) owner for deployment. (2) the code is processed by the administrative owner by injecting the interception code redirecting to the policy engine if needed. (3) the code is deployed including the engine if not present. (4) at run-time, the applications execute within the WSN subject to resource control with data flow as expected by the application developer. Calls to protected resources are intercepted, delegated to the policy engine and evaluated for access. In parallel, thus also at application run-time, the administrative owner (1) sets the policy that (2) gets disseminated to the enforcement engines in the network by some secure propagation mechanism. Since this policy definition is performed in the resource-rich backend environment, complex policy negotiations can be performed before the candidate policy is established, for example to resolve conflicts between competing applications of different actors. The end result is non-intrusive for developers and enforces the dynamic resource control objectives of the administrative owner as desired.

5 Research Prototype

Our validation strategy is twofold. We demonstrate the validity of injection in WSNs by demonstrating pre-deployment byte code weaving of the interception code using

AspectJ [11] on the Sun SPOT platform. Next, we present an implementation of the policy-driven security monitor engine to demonstrate feasibility of this critical TCB subsystem.

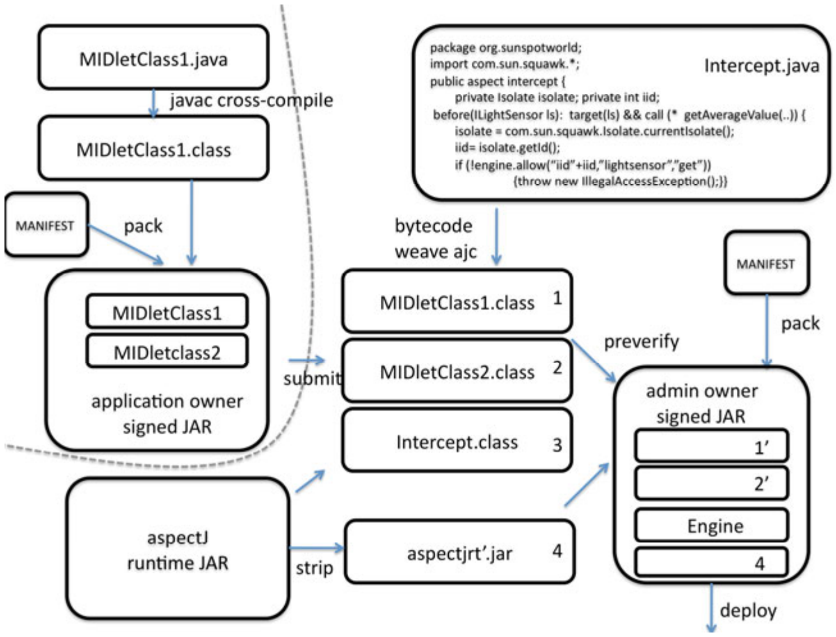
5.1 Pre-deployment Processing

During the pre-deployment phase, the administrative owner instruments the byte code of the application owner. Resource calls are captured through appropriate definitions of pointcuts, and advices specify the subsequent action, in this case redirection to the enforcement engine.

Our test platform is the SUN Spot - a J2ME compliant WSN platform implementing the MIDP profile [12] (180 MHz 32-bit ARM9, 512K RAM, SQUAWK VM 'Blue'). It already provides for safe concurrency through the isolate mechanism, a key requirement for security implementation. Multiple isolates run on a single virtual machine and are objects with multiple threads and associated state. For the SPOT version used for the tests, applications (MIDLets in the J2ME context, running within Sun SPOT isolates) are combined into a "jar" unit of deployment described in the manifest. The code injection uses standard tools, leveraging Java and AspectJ as opposed to custom rewriting of the byte code [13]. A largely similar approach can be followed for systems programmed in C. The resulting tool chain (Figure 2) is a combination of the AspectJ tools and Sun SPOT compilation, byte code verification and deployment utilities. Whilst the standard AspectJ runtime used for weaving is large, it can be stripped down significantly before deployment for inclusion in the deployment jar since the majority of the language features are not needed (such as cflow or reflection). The stripped AspectJ runtime is 2.3 kB, not including the engine code. So the use of aspect based instrumentation does not bring a significant code size increase and no further effort was done to optimize. Relating the figure to the stages of operational model described in Section 4.5:

1. The application owner *submits* the JAR containing the various MIDlets (`MIDletClass1,2`) to the administrative owner.
2. The administrative owner subsequently *unpacks* the JAR *and weaves* with his aspects (`Intercept.Java`).
3. The administrative owner *combines (preverifies/packs)* the instrumented MIDlets (`MIDletClass1', 2'`), the policy engine (`Engine`) and the stripped AspectJ runtime (`aspectjrt'.jar`) resulting in a new JAR. This JAR is then *deployed.*
4. The instrumented applications execute at run-time (not shown)

To process the resource access request, the engine needs at least the requester and resource information, but other context information may be included. The resource is available through the aspect definition, while the requester is tied to the application through the isolate-id and the manifest under control of the administrative owner, the same actor that decides on the policy.



```
package org.sunspotworld;
import com.sun.squawk.*;
public aspect intercept {
      private Isolate isolate; private int iid;
 before(ILightSensor ls):  target(ls) && call (*  getAverageValue(..)) {
      isolate = com.sun.squawk.Isolate.currentIsolate();
      iid= isolate.getId();
      if (!engine.allow("iid"+iid,"lightsensor","get"))
               {throw new IllegalAccessException();}}
```

Fig. 2. Tool chain (activities by application owner top-left, all others by administrative owner)

5.2 Policy-Driven Engine

The task of the policy engine is to enforce control on a particular resource, like access control or accounting. Upon interception of an access request, the policy engine evaluates the data on the requester and target resource by matching with the enabled control policies. These policies are to be defined by the node's administrative owner and, in case of access control, specify the operations allowed by a particular application component on a resource. In its simplest form, the outcome of the access control matching process is a permit or deny of the resource request. The policy engine collapses Policy Decision and Enforcement Point functions whilst the Policy Information Point resides in the backed.

Specification of resource control policies by the administrative owner uses rules following Event-Condition-Action (ECA) semantics. An ECA policy consists of a description of the triggering events, an optional condition which is a logical expression typically referring to the triggering events and external system aspects, and a list of actions to be enforced in response (see listing). Currently the engine itself does not keep state, but extensions to concepts such as access rates are feasible.

Not being strictly limited to security, since allowing for experimentation with other non-functional properties, our prototype currently follows a simplified version based on the Policy Description Language (PDL) semantics [14]. Upon specification by the administrative owner, the policies are automatically checked for consistency, parsed and transformed in the resource-rich backend into a binary data format for transport to the engine. For policy life cycle management, the engine provides a management API allowing for dynamic installation, removal and replacement of the policies.

```
policy "example access control" {
   on accessrequest req
   if(req.applicationID == "iid7" &&
      req.resourceId == "lightsensor" &&
      req.operationId == "get")
   then allow
}
```

Our prototype of a policy engine is implemented on Sun SPOT sensor nodes and currently requires 28 kB of memory. The policy engine offers support for evaluation of complex policy conditions consisting of a combination of both logical and mathematical operators. This gives the administrative owner of a node some freedom to specify additional constraints regarding resource access by including context. The current set of possible actions includes support for allowing and denying resource access. The binary representation of a single access policy item sent by the administrative owner backend takes up 142 bytes on average. After the policy distribution phase, during node-level commissioning, the policy engine's interpreter requires 6 ms on average to one-time transform the binary representation of a policy sent by the administrative owner backend into a data structure that is more suitable for efficient policy evaluation. This data structure is implemented as a Java object and consumes 420 bytes of RAM. Evaluating whether a requester can have access to a resource takes on average less than 2 ms (1 policy evaluation). These metrics clearly show the feasibility of policy-based access control in the real-world scenarios we focus on. More details on our policy language and performance can be found in [15].

6 Related Work

Several research projects (ITAIDE [16], INTEGRITY [17]), whilst focusing on the issues of the application domain (e-customs, global container tracking) highlight the multi-actor security problem and confirm the need for fine-grained security controls.

An overview of low level security solutions - many of which are needed to build TCBs - can be found in [6][18][19]. The available security research combines into more complete suites such as SPINS 'Security Protocols for Sensor Networks' or TinySec/MiniSec (when combined with a key distribution mechanism), focusing on data link layer security. These form the cornerstone of middleware security although individual services such as code distribution, routing, aggregation or localization can and are being individually optimized for security [10]. End-to-end security on our target platform (Sun SPOT) is realized using ECC asymmetric crypto [12].

More specifically we identify related work in the following fields: middleware, life cycle management, code instrumentation, aspect-oriented work on small platforms and policy-driven systems. Many middleware systems for WSNs focus on service re-use and providing higher-level services [7]. However, little middleware support is found for networks used by competing actors and applications. Systems such as MiLan [2] or AutoSec [20] recognize the separation between resources and application and the involved competition, and deploy or adjust accordingly to maintain quality guarantees. They provide an opportunity for global resource optimization at runtime but do not explicitly recognize the security context of today's settings. Other research has extended the middleware concepts beyond the runtime phase of the system life cycle. From a development perspective this touches on the work of middleware programming abstractions, such as database abstractions or macroprogramming [8]. Whilst elegant to express network-wide, functional objectives, fine-grained node-level security policies are hard to express in this context of high-level abstractions. RuleCaster [21] provides for logical specification of the application objective, but equally provides support for describing the physical (distribution) structure and the actual WSN infrastructure. Development, deployment and runtime are treated holistically in RuleCaster and the compilation and pre-deployment steps are key phases to weave these perspectives together. The presented evaluation method only validates the software engineering approach. By using the compiler step as a way to achieve energy-efficiency, Sadilek [22] validates the idea of looking beyond the runtime to achieve non-functional goals.

The t-kernel [13] provides operating system protection and preemption next to other features by binary translation at load time on MICA2 motes. This code instrumentation approach could be extended with resource protection enforcement by including a distributed policy-driven monitor. Tuohimaa and Leppänen [23] present a code security monitor idea using aspects for the J2ME platform for mobile phones disregarding the specifics of sensor networks. The focus is on code flow control analog to automata [5] rather than resource access. No implementation is provided – the proposed weaver is conceptual and feasibility not demonstrated. Walton and Eide [24] approach resource management by providing an aspect language-based based approach and extend NesC. The controlled resources are of a low-level system nature such as memory management or real-time considerations. The proposed system is not policy-based and the reader is referred to future work for implementation.

Platon and Sei [19] provide a survey of state-of-the-art in WSN security emphasizing the need for distributed policy-driven security to provide for scalability. A policy-based approach for implementing functional policies expressed as event-condition-action rules has been presented earlier for Body Area Networks [25]. The enforcement engine is available in various forms corresponding to capabilities of the various tiers. This work is being extended for non-functional policies such as security or quality-of-service. Policies are applied in ESCAPE [26] to govern interactions between components in sensor network applications communicating through a pub/sub approach. Policy enforcement is possible only at four points inside the pub/sub broker. Marsh et al. [27] provide for a flexible and memory efficient policy specification language (WASL) validating the policy-based approach for WSNs.

7 Open Issues and Conclusions

The proposed solution for resource control has two cornerstone concepts that make it specific to WSNs. First, optimizing for resource consumption must look at the complete system life cycle in an integrated way - development, pre-deployment, deployment and runtime all provide optimization opportunities. Second, policy-driven controls provide adaptability at runtime through the use of a control mechanism that can efficiently address the variations in sharing objectives.

Several questions remain. The pre-runtime phases always miss the dynamic, real-time view on the actual resource-application space so decisions taken during these phases clearly will be suboptimal. Further work is needed to assess true savings of pre-deployment processing beyond it being a suitable point to redirect to runtime decision constructs. For example, if by code verification it is already possible to flag illegal resource accesses – why bother to deploy? Also more effort is needed to find the exact equilibrium between increasing control functionality and resource consumption in real-world scenarios. Which concerns need to be policy-driven, and to what extent? Also, for many non-functional qualities the line between application logic and policy is not a clear cut, which complicates the approach.

In future WSN settings resource sharing will often be an important factor and the need for controls regarding this functionality is key. In this paper, we presented a solution for resource security applicable to problems ranging from resource consumption accounting to effectively controlling real-time access. The presented solution based on targeted monitor injection and a non-intrusive operational model, with a maximum of tasks concentrated in the resource-rich backend, respects the restricted nature of the target environment. At run-time, the dynamics of the infrastructure and changes in goals of actors are accommodated through a policy-centric approach characterized by lightweight updates. A prototype is provided validating the feasibility of injection and policy-based resource security on WSNs.

Acknowledgments. Funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, IBBT and the Research Fund K.U. Leuven.

References

1. Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.: Wireless sensor networks for habitat monitoring. In: 1st ACM International Workshop on Wireless Sensor Networks and Applications, pp. 88–97. ACM, Atlanta (2002)
2. Heinzelman, W.B., Murphy, A.L., Carvalho, H.S., Perillo, M.A.: Middleware to support sensor network applications. IEEE Network 18, 6–14 (2004)
3. Huygens, C., Joosen, W.: Federated and Shared Use of Sensor Networks through Security Middleware. In: Sixth International Conference on Information Technology: New Generations, Las Vegas, NV, USA, pp. 1005–1011 (2009)
4. Anderson, J.P.: Computer Security Technology Planning Study. Hanscom AFB (1972)
5. Ligatti, J., Bauer, L., Walker, D.: Edit automata: enforcement mechanisms for run-time security policies. International Journal of Information Security 4, 2–16 (2005)
6. Walters, J.P., Liang, Z., Shi, W., Chaudhary, V.: Wireless Sensor Network Security: A Survey. In: Security in Distributed, Grid, and Pervasive Computing, ch. 17. CRC Press, Boca Raton (2006)
7. Wang, M.M., Cao, J.N., Li, J., Dasi, S.K.: Middleware for wireless sensor networks: A survey. Journal of Computer Science and Technology 23, 305–326 (2008)
8. Newton, R., Morrisett, G., Welsh, M.: The regiment macroprogramming system. In: 6th International Conference on Information Processing in Sensor Networks, pp. 489–498. ACM, Cambridge (2007)
9. Westhoff, D., Girao, J., Sarma, A.: Security Solutions for Wireless Sensor Networks. NEC Tech. J. 1, 106–111 (2006)

10. Deng, J., Han, R., Mishra, S.: Secure code distribution in dynamically programmable wireless sensor networks. In: 5th International Conference on Information Processing in Sensor Networks, pp. 292–300. ACM, Nashville (2006)
11. The AspectJ Project, `http://www.eclipse.org/aspectj`
12. SunSPOTWorld – Home, `http://www.sunspotworld.org`
13. Gu, L., Stankovic, J.A.: t-kernel: providing reliable OS support to wireless sensor networks. In: 4th International Conference on Embedded Networked Sensor Systems, pp. 1–14. ACM, Boulder (2006)
14. Lobo, J., Bhatia, R., Naqvi, S.: A policy description language. In: Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence, pp. 291–298. American Association for Artificial Intelligence, Orlando (1999)
15. Matthys, N., Hughes, D., Michiels, S., Huygens, C., Joosen, W.: Fine-Grained Tailoring of Component Behaviour for Embedded Systems. In: Lee, S., Narasimhan, P. (eds.) SEUS 2009. LNCS, vol. 5860, pp. 156–167. Springer, Heidelberg (2009)
16. Van Stijn, E., Bjorn-Andersen, N., Razmerita, L., Henriksen, H.: Improving International e-Customs–The European ITAIDE Initiative. In: First International Conference on the Digital Society (ICDS 2007), p. 21. Guadeloupe, French Caribbean (2007)
17. Integrity, `http://www.integrity-supplychain.eu`
18. Sabbah, E., Majeed, A., Kang, K., Liu, K., Abu-Ghazaleh, N.: An application-driven perspective on wireless sensor network security. In: 2nd ACM International Workshop on Quality of Service & Security for Wireless and Mobile Networks, pp. 1–8. ACM, Terromolinos (2006)
19. Platon, E., Sei, Y.: Security software engineering in wireless sensor networks. Progress in Informatics 5, 49–64 (2008)
20. Han, Q., Venkatasubramanian, N.: Information Collection Services for QoS-Aware Mobile Applications. IEEE Transactions on Mobile Computing 5, 518–535 (2006)
21. Bischoff, U., Kortuem, G.: Life cycle support for sensor network applications. In: 2nd International Workshop on Middleware for Sensor Networks, pp. 1–6. ACM, Newport Beach (2007)
22. Sadilek, D.A.: Energy-aware compilation for wireless sensor networks. In: 2nd International Workshop on Middleware for Sensor Networks, pp. 25–30. ACM, Newport Beach (2007)
23. Tuohimaa, S., Leppänen, V.: A compact aspect-based security monitor for J2ME applications. In: 2007 International Conference on Computer Systems and Technologies, pp. 1–6. ACM, Bulgaria (2007)
24. Walton, S., Eide, E.: Resource management aspects for sensor network software. In: 4th Workshop on Programming Languages and Operating Systems, pp. 1–5. ACM, Stevenson (2007)
25. Keoh, S., Twidle, K., Pryce, N., Lupu, E., Schaeffer Filho, A., Dulay, N., Sloman, M., Heeps, S., Strowes, S., Sventek, J.: Policy-based Management for Body-Sensor Networks. In: 4th International Workshop on Wearable and Implantable Body Sensor Networks (BSN 2007), IFMBE Proceedings, vol. 13, pp. 92–98 (2007)
26. Russello, G., Mostarda, L., Dulay, N.: ESCAPE: A Component-Based Policy Framework for Sense and React Applications. In: Chaudron, M.R.V., Szyperski, C., Reussner, R. (eds.) CBSE 2008. LNCS, vol. 5282, pp. 212–229. Springer, Heidelberg (2008)
27. Marsh, D.W., Baldwin, R.O., Mullins, B.E., Mills, R.F., Grimaila, M.R.: A security policy language for wireless sensor networks. J. Syst. Softw. 82, 101–111 (2009)