Wireless Sensor Network, 2011, 3, 10-17
doi:10.4236/wsn.2011.31002 Published Online January 2011 (http://www.SciRP.org/journal/wsn)

Concealed Integrity Monitoring for Wireless Sensor Networks

Björn Stelte, Thomas Bühring
Institut für Technische Informatik, Universität der Bundeswehr München, Neubiberg, Germany
E-mail: {*bjoern.stelte, thomas.buehring*}*@unibw.de*
Received December 21, 2010; *revised December* 29, 2010; *accepted January* 5, 2011

Abstract

Nowadays, sensor networks are widely installed around the world. Typical sensors provide data for healthcare, energy management, environmental monitoring, etc. In the future sensors will become a part of critical infrastructures. In such a scenario the network operator has to monitor the integrity of the network devices, otherwise the trustworthiness of the whole system is questionable. The problem is that every integrity protocol needs a secure channel between the devices. Therefore, we will introduce a covert channel for hidden transportation of integrity monitoring messages. The covert channel enables us to hide integrity check messages embedded into regular traffic without giving potential attackers a hint on the used integrity protocol.

Keywords: Covert Channel, Integrity Monitoring, Wireless Sensor Network

1. Introduction

A possible scenario for a distributed system that needs close monitoring of its integrity is a wireless sensor network (WSN). Aside from secure communication channels, the main concern in WSN is the integrity of nodes, since the widespread use of shared secrets, if any, in communication protocols is based on the assumption that nodes are not compromised.

Integrity of nodes may for example be verified by an attestation protocol, which uses a trusted platform module (TPM) to attest the integrity of cluster heads [1]. But in order to enable a WSN-operator to react to tampering attempts, information about node integrity needs to be exchanged throughout the network. If such information is exchanged overtly, attackers may be aware of the fact that the network is being monitored. Analysis of exchanged information may even reveal how often such information is exchanged and if no appropriate cryptographic countermeasures are taken, it may also be possible to tell what information is exchanged.

As a solution, the integrity monitoring information could be covertly embedded into traffic that fits the well-known purpose of the WSN, e.g., measurements of environment temperature. This does not only prevent attackers from determining what information is exchanged in order to assess node integrity, but it will even be difficult if not impossible to tell whether the integrity of a particular network is actually being monitored. Plain encryption of the communication channel in contrast does not provide a similar level of deception, since it may still be possible to distinguish ordinary traffic from integrity monitoring traffic, e.g., by tampering nodes while monitoring the change of message frequency or size.

2. Requirements

According to [2], the "quality of the covert channel can be expressed in terms of detectability (arbitration must be measurable by the recipient), indistinguishableness (hidden data cannot be separated from the cover information) and bandwidth (ratio of hidden data to cover data)." Yet, a covert channel for integrity monitoring must not only be of sufficient quality in terms of this definition but it must also meet some specific requirements derived from surveillance scenarios.

1) Delay: Typically, covert channels have a comparable low ratio of hidden data to cover data. In addition to that, they rely on the occurrence of network traffic that can be used as a disguise for the hidden communication. But to provide the means for efficient administrative reaction, a reasonable maximum delay for event notifications is required. In case of covert channels, signaling a critical system event would rely on ordinary cover traffic taking place. A general question is, what can be done if

there is not enough cover traffic or no cover traffic at all when a critical event needs to be signaled. Regarding scenarios such as monitoring critical infrastructure or even in military scenarios, continuous and sufficient network traffic are assumed.

2) Recurrence: If a covert channel is used to repeatedly communicate possibly unchanged information of a node, care must be taken to avoid recurring patterns when embedding this information into traffic that would otherwise not have recurring patterns. The challenge is therefore not only to hide occasional, differing event notifications but also to disguise the transmission of recurring events or health information.

3) Unidirectional channel: For the purpose of this discussion only a unidirectional channel is required, even if WSNs in surveillance and military scenarios would benefit from a bidirectional communication channel for delivering sensor control commands. This is because for the sake of simplicity the focus of this work is narrowed and the results from a unidirectional communication channel can possibly be transferred to a bidirectional situation.

3. Related Work

Covert channels were first defined by Lampson as channels "not intended for information transfer at all, such as the service program's effect on the system load" [3]. Covert channels can be classified in the broader scope of information hiding. However, there is no clear distinction between steganography and covert channels. A possible explanation is that covert channels establish information flows between entities which would otherwise not be allowed to communicate at all or using channels that are not intended for communication, while steganography enables entities to convey additional information among legitimate, or disclosed, data. The term "covert channels" seems to have prevailed in the context of networks and is used to describe any information hiding approaches in network communications, sometimes merrily mixed with "steganography".

Simmons *et al.* [4] introduced an illustrative, yet formal setting for the steganographic problem: Two prisoners (e.g., Alice and Bob) are only allowed to communicate through a warden (e.g., Willie). Their goal is to devise an escape plan without the warden noticing while the warden hopes to deceive them by altering communication or introducing bogus messages. Further, active wardens, which might deliberately alter or introduce messages, and passive wardens, which are limited to observing communication, are distinguished [5].

Lampson [3] defined covert channels in the context of process isolation within a single host operating system. Today the possibility of covert channels in distributed systems is a major concern, e.g. if organizations intend to control particularly outbound information flows from the organization's network.

In practice, two types of covert channels can be distinguished by their realization [6]. Storage-based covert channels involve the direct or indirect reading and writing of storage locations between different processes and timing-based channels use an modulation of system resources that can be observed by another process. Apparently, covert channels can also be characterized by the layer(s) of the OSI reference model on which they operate. Handel and Sandford [2] demonstrated, how each OSI layer could be used as a basis for implementing a covert channel.

The evaluation of covert channels is only possible with regard to a specific deployment. Still, it is assumed that protocol header manipulation will deliver better performance than payload steganography. At the same time, a protocol header manipulation algorithm for a communication protocol introduces less requirements on the particular type of cover traffic compared to applying steganography to the cover payload. Hence, this section is focused on covert channels based on network header manipulation.

4. Model of a Network Covert Channel for Integrity Monitoring (NCCIM)

This section develops a model of a unidirectional channel that connects a sensor agent application and monitoring application, which are running on different network nodes. This model of a Network Covert Channel for Integrity Monitoring (NCCIM) is intended to be a modular framework for providing robust communication between sensor and monitor. The framework is based on the OSI reference model to provide flexibility for implementations.

The sending stack of the NCCIM is responsible for embedding the covert traffic on a node that is monitored by a sensor agent. It interfaces with the sensor agent application and the cover traffic source, which is the subsystem handling network traffic generated by other services running on that node. Similarly, the NCCIM receiving stack extracts the covert monitoring information on the node running the monitoring application. It interfaces with the cover traffic sink, which is the subsystem responsible for receiving network traffic of that node, and the monitoring application. The NCCIM model provides a framework for the modification and interception of cover traffic. Monitoring information is covertly communicated from sensors to a monitor. The covert channel is developed given the following assumptions about the nodes running the sensor agent or monitoring agent:

1) The overt services running on the node being monitored generate a continuous traffic flow that provides an upper bound on the delay of the covert channel.

2) The subsystem of the nodes responsible for delivering network traffic (traffic source) and the subsystem responsible of receiving network traffic (traffic sink) allow traffic to be modified and to be intercepted, respectively.

3) The nodes running the sensor agents are based on an execution model that supports concurrency.

4) Each sensor agent reports to the same monitor.

NCCIM is not responsible for gathering monitoring data, which is performed by the sensor agents, or analyzing such data, which is the task of the monitoring agent. However, the covert channel must ensure some requirements on the monitoring information being transmitted. Given the characteristics of a covert channel as the limited capacity and the need for hiding any communication associations, the relationship between sensor agents and monitoring agents should be managed by the NCCIM. In addition to that, ordinary communication tasks from transport to physical layer need to be performed. Hence, the NCCIM must operate on OSI layers 1 (Physical Layer) to 6 (Session Layer). For further discussion, problems that can be solved in a more or less platform-independent way are assigned to different entities based on the OSI-reference model.

Monitoring information gathered by the sensor agents can be represented in various formats. Well-known standard formats include syslog messages or the RMON MIB. Custom formats may be based on XML, but it is also possible to use an optimized binary format. The actual representation of the monitoring information is therefore out of the scope of this architecture. However, it is relevant if all messages are of fixed size or if variable length messages must be handled (Pa6.1, see **Table 1**). Regardless of the fact whether messages have variable size, the (maximum or general) message length determines if fragmentation is necessary at the Data Link Layer (Pa6.2). Messages smaller than the maximum message size can be padded (Pa6.3).

Table 1. Presentation layer parameters.

ID	Name	Description
Pa6.1	Fixed message size	Boolean value that determines, if monitoring information is represented in fixed size messages.
Pa6.2	Maximum message size	Unsigned integer value denoting the maximum size of monitoring information to be transmitted. If fixed size messages are used or messages are padded, this is the size of all messages.
Pa6.3	Message padding	Boolean value that is true if a padding is appended to messages smaller than the maximum message size. Is only used if the application does not provide fixed size messages.

Table 2. Session layer interfaces.

ID	Primitive	Parameters
Sensor Session Entity		
Pr5.1	Sensor Agent Up	Identifier of the node that is started and optional information payload.
Pr5.2	Sensor Agent Down	Identifier of the node that is stopped and optional information payload.
Pr5.3	Sensor Agent Information	Identifier of the node sending the information and the information payload to be delivered.
Monitor Session Entity		
Pr5.1	Received Up Information	Node identifier and optional information payload.
Pr5.2	Received Down Information	Node identifier and optional information payload.
Pr5.3	Received Monitoring Information	Node identifier and monitoring information payload.

4.1. Managing the Relationship Between Sensor Agent and Monitoring Agent

The *Sensor Session Entity* provides three access points to the sensor agent application (see **Table 2**). The *Sensor Agent Up* and *Sensor Agent Down* primitives indicate that the sensor agent is started and stopped, respectively. The *Sensor Agent Information* primitive is used to send event or status information.

The *Monitor Session Entity* requires according upper layer interfaces, to inform the monitor application about *Sensor Association Created*, *Sensor Association Deleted* and to deliver monitoring messages (*Sensor Information Received*). In addition to that, the *Sensor Association Timeout* primitive is used to notify the monitor application if a sensor agent failed to send a keep-alive message within the expected time range.

On receipt of a *Sensor Agent Up* primitive, the *Sensor Session Entity* creates an association between the given sensor agent node identifier and the monitor node identifier (in **Table 3**), which is known beforehand. This association is created by invoking the *Send Up Information* primitive of the *Sensor Transport Entity* with the sensor agent node identifier. If the *Monitor Session Entity* receives an *Up Information Received* primitive, the association is created at the monitor. A similar procedure is used for the *Sensor Agent Down* primitive, which deletes the association. The *Sensor Agent Information* primitive is mapped to the *Send Monitoring Information* primitive. Node registration and authentication are out of scope of this architecture and should be provided by the sensor and monitor application, respectively.

Table 3. Application layer interfaces.

ID	Primitive	Parameters
Monitor Application Entity		
Pr7.1	Sensor Association Created	*Sensor agent node identifier* for which the association was created.
Pr7.2	Sensor Association Deleted	*Sensor agent node identifier* for which the association was deleted.
Pr7.3	Sensor Information Received	*Sensor agent node identifier* for which information was received and the *information payload*.
Pr7.4	Sensor Association Timeout	*Sensor agent node identifier* for which no keep-alive message was received within the expected time interval.

If keep-alive messages are used, the session layer maintains a timer for each association scheduling the next keep-alive message. The timer is initialized during the creation of the association and is set to the keep-alive interval (Pa5.1) minus a pseudo-random offset. If a keep alive message is due, the *Send Monitoring Information* primitive is invoked for the particular association with an optional additional payload provided by the sensor application (which can be used for latency measurement if clocks are synchronized) and the timer is re-initialized as described above. If a monitoring message is sent for this association, the timer for the next keep-alive message is reset to its initial value. Accordingly, the *Monitor Session Entity* keeps track of all associations and the latest valid arrival time for a keep-alive message. If this time is exceeded, the *Sensor Association Timeout* primitive of the monitor application is invoked. The latest valid arrival time is set on creation of the association and on receipt of a *Received Monitoring Information* primitive from the *Monitor Transport Entity*. It is calculated from the current time, the keep-alive interval (Pa5.1, as shown in **Table 4**) and an implementation dependent tolerance offset (Pa5.2).

4.2. Transport Layer

As shown in **Table 5**, the *Sensor Transport Entity* provides the access points *Send Up Information*, *Send Down Information* and *Send Monitoring Information* to the *Sensor Session Entity*. Analogous to the sensor, the *Monitor Transport Entity* issues the primitives *Received Up Information*, *Received Down Information* and *Received Monitoring Information*. The task of the transport layer in this architecture basically is to perform integrity checks (Pa4.3).

The channel provided by this architecture is unidirectional and thus unreliable, which avoids the complexity

Table 4. Session layer parameters.

ID	Name	Description
Pa5.1	Keep-alive interval	Unsigned integer value indicating the maximum time between two keep-alive messages (in seconds). If zero, sending entities do not verify the covert channel availability by periodically sending keep-alive messages.
Pa5.2	Keep-alive tolerance	Unsigned integer value giving an implementation dependent tolerance for the keep-alive interval.

Table 5. Transport layer interfaces.

ID	Primitive	Parameters
Sensor Transport Entity		
Pr4.1	Send Up Information	Identifier of the node that is started and data to be sent, which is the data header (including node identifier) and an optional monitoring payload.
Pr4.2	Send Down Information	Identifier of the node that is stopped and data to be sent, which is the data header (including node identifier) and an optional monitoring payload.
Pr4.3	Send Monitoring Information	Identifier of the node that sends the monitoring information and data to be sent, which is the data header (including node identifier) and the monitoring payload.
Monitor Transport Entity		
Pr4.1	Received Segment	Node identifier and the segment, which may be constructed from several frames.

of a bidirectional protocol and reduces requirements that must be met by an underlying communication platform. However, loss of messages can be detected by the receiver if both, sender and receiver, use a consistent segment (sequence) numbering scheme (Pa4.2). The initial sequence number for a new association is when the *Sensor Transport Entity* processes a *Send Up Information* primitive. The algorithm to determine the initial sequence number from the node identifier and other suitable characteristics known by the sensor as well as the monitor is implementation dependent and may be based on a shared secret, which acts as a seed. A non-trivial assignment of initial sequence numbers enables the monitor to authenticate the process of creating an association, as the result of this check is included in the Received Up Information primitive.

The MIC value is calculated by an implementation dependent function. The input for the MIC calculation consists of segment header, where the MIC field is set to all zeros, and segment body.

If a *Received Segment* primitive is issued by the Monitor Data Link Entity for a node identifier for which no

proper Up segment was received before, the *Received Up Information* primitive is issued prior to the *Received Monitoring Information* primitive. This behavior accounts for the unidirectional channel and will further be referred to as implicit association creation.

The distinction between up, down and message segments is introduced for the sole purpose of managing the process of assigning sequence numbers and the associated resources. The segment is composed from the segment header, which consists of the segment type field (Pa4.1, **Table 6**) and the sequence number, if any, and the segment body containing the monitoring information.

Because of the absence of multiple monitors (assumption 4), no logical addressing of recipients is required. Furthermore, the realization of multi-hop communication using the covert channel is highly implementation dependent as explained below. Hence, introducing a network layer is regarded as an unnecessary overhead and segments are directly interchanged with the Data Link Layer.

If possible at all, multi-hop communication with the monitor using a covert channel can be realized using different approaches. Flooding may be implemented by embedding the covert channel in traffic to all communication partners. Static upstream routes can be represented as filters, selecting which PDU of the overt channel are used for embedding the covert information. This decision may for example be based on the destination address of a PDU and enables the exploitation of more complex routes of the overt channel. Such procedures belong to the physical layer of the covert channel, which is implementation dependent.

4.3. Encoding into Cover Traffic

The data link layer and physical layer are responsible for embedding the covert channel into overt traffic. While most aspects of the physical layer depend on the specific implementation used, the concepts of the data link layer can be formulated in a generic way. As shown in **Table 7**, the *Sensor Data Link Entity* provides the *Push Segment* access point to the *Sensor Transport Entity* and the *Pop Frame* access point to the *Sensor Physical Entity*, while the *Monitor Data Link Entity* issues the *Received Segment* primitive to the *Monitor Transport Entity*.

Table 6. Transport layer parameters.

ID	Name	Description
Pa4.1	Segment Type Size	Unsigned integer giving the size of the segment type field in bits.
Pa4.2	Sequence Number Size	Unsigned integer value indicating the size of sequence numbers. If zero, no sequence numbers are sued.
Pa4.3	Message integrity size	Unsigned integer value which is zero if no message integrity check is performed.

Table 7. Data link layer interfaces.

ID	Primitive	Parameters
Sensor Data Link Entity		
Pr2.1	Push Segment	*Identifier of the node* for which the segment is sent and the *segment to be sent*, which may require fragmentation.
Pr2.2	Pop Frame	(no parameters)
Monitor Data Link Entity		
Pr2.1	Push Frame	A buffer containing the received *frame*.

These entities performs the task of (de-)fragmentation, if the sum of the maximum size of monitoring information (Pa6.2) and the size of all headers is larger than the maximum frame size (Pa2.2, **Table 8**). An implementation determines the frame size from the maximum number of bits that can be embedded in a cover PDU. If fragmentation is required, it can be necessary to prefix the frame not only with a node identifier (Pa2.1) but also with a fragment number (Pa2.3). The prefix is then used to determine the right ordering of the frames to get the original segment. It is only required if the protocol used for traffic encoding does not guarantee sequential delivery of data (for example IP) or if the traffic interception is done in such a ways that the or dering features of a sequential protocol cannot be used (for example intercepting TCP traffic in the systems IP stack). The number of bits available for representing the fragment number (Pa2.4) limits the number of possible fragments. Thus, it should be consistent with the maximum data size (Pa6.2), if not, errors may be introduced.

Table 8. Data link layer parameters.

ID	Name	Description
Pa2.1	Node identifier size	Unsigned integer value denoting the size of node identifiers.
Pa2.2	Frame size	Unsigned integer giving the number of bits to be sent in one block. Determines if fragmentation is necessary.
Pa2.3	Fragment numbering required	Boolean value indicating if fragments must be numbered.
Pa 2.4	Fragment number size	Unsigned integer indicating the number of bits available for representing the frame number.
Pa 2.5	Maximum frame delay	Unsigned integer value giving a maximum time for a frame to be sent (in seconds).
Pa 2.6	Frame delay tolerance	Unsigned integer providing an implementation dependent monitor side tolerance offset.

The *Sensor Data Link Entity* maintains a queue of frames to be sent (*frame queue*). The entries of this queue consist of the frame and the latest valid encoding time. The latest valid encoding time is calculated from the time when the first frame of the current segment got pushed into the queue plus a maximum encoding delay (Pa2.5). If this time is exceeded, the whole segment is invalidated and the *Segment Encoding Timeout* primitive is issued, which is handed through the layers to the sensor application.

The actual encoding of bits is done by an implementation specific function (for example a packet injection hook) that retrieves a frame from the *frame queue* and encodes it into the cover traffic.

4.4. Channel Characteristics

Several characteristics useful for the evaluation of NCCIM implementations can be formulated. First, the overhead $s_{overhead}$ for a particular monitoring message *m* in bits is a linear function of the number of fragments $n_{frag}(m)$ required to encode *m*, the size of a frame header $s_{overhead}$ and the size of a segment header $s_{segmenthdr*}$.

$$s_{overhead} = n_{frag}(m)\, s_{framehdr} + s_{segmenthdr}$$

While the message size |*m*| and thus the number of fragments may be variable, the other values are implementation dependent constants derived from the size of the different header fields indicated by *s*. In particular, S_{frame} is the total size of a frame and as described below, it depends on the encoding function used.

$$n_{frag}(m) = \left\lceil \frac{|m|}{S_{frame} - S_{framehdr}} \right\rceil$$

$$s_{framehdr} = s_{fragnum} + s_{nodeid}$$

$$s_{segmenthdr} = s_{segtype} + s_{seqnum} + s_{mic}$$

Given an encoding function *c* that encodes a secret frame into a cover-PDU *x*, the ratio between the cover-PDU size |*x*| and the number of secret bits encoded by *c* can be expressed as

$$r_c(x) = \frac{|x|}{n_{encoded,c}(x)}$$

For the majority of observed covert channel implementation, $n_{encoded,c}$ is a constant, *i.e.*, the number of bits encoded does not depend on the particular cover-PDU. This is assumed for all NCCIM implementations. Furthermore, a frame is expected to be the smallest bit sequence that can be transmitted in one piece. In this case, $n_{encoded,c} = S_{frame}$.

The number of cover-PDU required to encode a monitoring message *m* is therefore $n_{frag}(m)$. However, the above characteristics can only be expressed with respect to secret and overt network traffic samples. Thus, a general evaluation of a NCCIM implementation is possible, if corresponding average values are known or assumed. It is then possible to express the time required to transmit *m* as

$$\hat{t} = \overline{n}_{frag} \times \overline{t}_{cover}$$

where $\overline{t_{cover}}$ is the average inter-arrival time of cover-PDU.

5. Prototypical Implementation for Wireless Sensor Networks

In this section, a very basic implementation of the scenario of a WSN covert channel is presented. The overt task of the WSN deployed in this implementation (see **Figure 1**) is to collect temperature information. To protect the network against theft of sensor nodes, a covert channel is used to alert the operator if a node is moved. This prototypical implementation is focused on 1) illustrating the utility of a covert channel for hidden network supervision and 2) demonstrating the feasibility of the infrastructural requirements of the NCCIM physical layer in a WSN deployment. Hence, this covert channel does not provide strong protection against detection or modification of the covert information.

5.1. Hardware

The implementation is based on Crossbow MICAz MPR2600 motes, which are a retail version of the MICAz MPR2400. The following information is based on the description of the MPR2400 mote in [7]. These motes use a IEEE 802.15.4 compliant Chipcon CC2420 radio and are operated using a Atmega128L micro-controller. For the temperature and acceleration measurements, the Crossbow MTS400CA sensor board [8] is added. The particular sensors used are the Sensirion SHT11 (temperature) and the Analog Devices ADXL202JE (acceleration). The motes were programmed using the Crossbow MIB520 USB Interface Board.

5.2. Implementation Overview

The actual implementation is realized as software based on TinyOS 2.1 and the BLIP. It consists of a mote program, which measures temperature and detects node movement by acceleration changes, and two distinct programs running on the gateway host (see **Figure 1**).

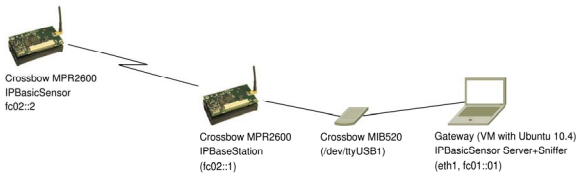

Figure 1. Overview of the wireless sensor network setup.

Overtly, only the temperature and an internal counter value are sent to the gateway using UDP packets. The information about whether the node was moved or not is encoded into a rudimentary covert channel. The IPBaseStation program and the ip-driver used to connect the WSN to the host-only IPv6 network are provided by the BLIP software, which is included in TinyOS 2.1.

5.3. Sensor Node Program

The IPBasicSensor program running on the sensor nodes performs the measurement tasks and reports to the gateway host using UDP. It is written in NesC and uses several components provided by TinyOS 2.1. Temperature and acceleration are measured by the program in intervals of 250 milliseconds. The temperature is stored for later use, while the current acceleration is compared to that measured before. If the difference between both values is greater than a predefined tolerance, a node movement is detected. Because the program is only intended to be a proof of concept implementation, the false positives or negatives resulting from this simple algorithm are acceptable. Every five seconds, a UDP datagram containing a measurement report is sent to the gateway host. A modified BLIP stack is used to send the datagrams, which implements the encoding of covert information into the overt measurement reports.

Among other, the program uses the UDPShellC component, which is also included in TinyOS 2.1, to provide a simple CLI that can be accessed from IPv6 hosts using UDP. The CLI is used for debugging purposes, because it provides diagnostic utilities like a ping-command or custom commands displaying internal data structures. In addition to that, the predefined tolerance for the movement detection can be adjusted using the CLI.

5.4. Gateway Server and Sniffer

In the implementation setup, there are two programs intended to run on the server. basic_sensor_server is performs the overt sensing task of the example WSN. The program creates a socket, which binds to the UDP port specified in the header file. As long as it is not interrupted by the user, it receives the measurement reports sent by the sensor node and prints these reports to the command line.

basic_sensor_sniffer can be used to decode the covert information from the traffic. The sniffer is using the PCAP library to capture IPv6 traffic on the specified interface. It displays a text message if the cover traffic contains the information that a node movement was detected by IPBasicSensor.

6. Covert Channel

Because this implementation serves solely for demonstration purposes, it does not implement the layer model from Section 4. Instead, a basic covert channel data link layer is sufficient because the channel is only used to transmit an indication, whether the sensor node was moved or not. On the covert channel physical layer, this information is encoded in the IPv6 hop limit header field. For the sake of simplicity, the overt channel in this sensor node example involves significant overhead regarding the use of IPv6 and UDP headers compared to 32 bit sensor payload. In contrast, the covert channel introduces no overhead, as the 8 bit frame is directly written to the hop limit field, which is of the same size. This is why the covert channel is easy to detect. Besides, it does not provide integrity check and it does not enable loss detection or the tracking of associations between sensor and monitor. This illustrates why a robust covert channel based on the model presented in Section 4 provides limited capacity while requiring significant overhead at the same time.

7. Conclusions

As the presented proof-of-concept prototype in Section 5 illustrates, network covert channels can be used to hide the integrity monitoring of a distributed WSN system. If implemented properly, a covert channel does not merely disguise that the distributed system is supervised, it also avoids disclosing what event or status information is transmitted and, given sufficient and continuous cover traffic, in which intervals this is done.

The actual encoding of secret fragments into cover traffic is subject to extensive research, which is to some extent presented in Section 3. However, a deployment of a covert channel in the context of integrity monitoring requires a certain degree of robustness, which is highly dependent on the deployment platform and the particular type of cover traffic. This characteristic is not provided by the majority of the proof of concept implementations.

The model of a network covert channel presented in Section 4 gives a modular description of the necessary tasks to ensure robust covert communications between sensor nodes and a monitor. The layer architecture of the model provides flexibility for specific requirements of

different deployment platforms or different types of cover traffic. From the description of several channel characteristics it can be seen that there is a trade-off between robustness and capacity of a covert channel.

8. References

[1] C. Krauß, F. Stumpf and C. Eckert, "Detecting Node Compromise in Hybrid Wireless Sensor Networks Using Attestation Techniques," *Proceedings of the 4th European conference on Security and Privacy in Ad-Hoc and Sensor Networks*, 2007.

[2] T. G. Handel and M. T. Sandford, "Hiding Data in the OSI Network Model," *Proceedings of the First International Workshop on Information Hiding*, Vol. 1174, 1996, pp. 23-38. doi: 10.1007/3-540-61996-8_29

[3] B. W. Lampson, "A Note on the Confinement Problem," *Communications of the ACM*, Vol. 16, No. 10, 1973. doi:10.1145/362375.362389

[4] G. J. Simmons, "The Prisoner's Problem and the Subliminal Channel," *Workshop on Communications Security* (*CRYPTO*'83), 1984.

[5] F. A. P. Petitcolas, R. J. Anderson and M. G. Kuhn, "Information Hiding — A Survey," *Proceedings of the IEEE, Special Issue on Protection of Multimedia Content*, 1999.

[6] National Computer Security Center, "A Guide to Understanding Covert Channel Analysis of Trusted Systems," 1993.

[7] Crossbow Technology Inc., "MPR-MIB Users Manual," June 2006.

[8] Crossbow Technology Inc., "MTS/MDA Sensor Board Users Manual," June 2006.

