Tampering with Motes: Real-World Physical Attacks on Wireless Sensor Networks

Alexander Becher, Zinaida Benenson, and Maximillian Dornseif

RWTH Aachen, Department of Computer Science,
52056 Aachen, Germany
alex@cyathus.de, zina@i4.informatik.rwth-aachen.de,
dornseif@informatik.uni-mannheim.de

Abstract. Most security protocols for wireless sensor networks (WSN) assume that the adversary can gain full control over a sensor node through direct physical access (node capture attack). But so far the amount of effort an attacker has to undertake in a node capture attack is unknown. In our project we evaluate different physical attacks against sensor node hardware. Detailed knowledge about the effort needed for physical attacks allows to fine tune security protocols in WSNs so they provide optimal protection at minimal cost.

1 Introduction

Wireless sensor networks (WSN) consist of a large amount of *sensor nodes*, which are small low-cost wireless computing devices equipped with different sensors. They gather and process environmental data like temperature, humidity, light conditions, seismic activities. Sensor nodes are also sometimes called *motes* because of their small size and the envisioned deployment pattern: They are supposed to be spread over a large geographic area, organize themselves into an ad hoc network, and operate unattended for months or even years. Many intended applications, such as military surveillance, but also structural health monitoring (detecting damage in buildings, bridges, aircrafts), supply chain management, or building protection, have high security requirements. For an overview of security issues in sensor networks, see [23, 24].

One of possible attacks on WSNs is called *node capture*. This describes a scenario where an adversary can gain full control over some sensor nodes through direct physical access. As sensor nodes operate unattended and cannot be made tamper proof because they should be as cheap as possible, this scenario is more likely than in most other computing environments. This type of attack is fundamentally different from gaining control over a sensor node remotely through some software bug. As all sensors are can be assumed to run the same software, finding an appropriate bug would allow the adversary to control the whole sensor network. In contrast, a node capture attack can only be mounted on a small portion of a sufficiently large network.

Depending on the WSN architecture, node capture attacks can have significant impact. Thus, most existing routing mechanisms for WSNs can be substantially influenced even through capture of a minute portion of the network [14].

In the TinySec mechanism [13], which enables secure and authenticated communication between the sensor nodes by means of a network-wide shared master key, capture of a single sensor node suffices to give the adversary unrestricted access to the WSN.

Most current security mechanisms for WSNs take node capture into account. It is usually assumed that node capture is "easy". Thus, some security mechanisms are verified with respect to being able to resist capture of 100 and more sensor nodes out of 10,000 [6, 12]. However, to the best of our knowledge, nobody ever tried to determine the actual cost and effort needed to attack currently available sensor nodes.

Thus we set out to verify the assumption that node capture is easy. The contributions of this work are as follows:

1. We developed a design space for *physical attacks* on sensor nodes. These are all attacks that require having direct physical access to the sensor nodes.
2. We found out that node capture attacks which give the adversary full control over a sensor node are not so easy as usually assumed in the literature. They require expert knowledge, costly equipment and other resources, and, most important, removal of nodes from the network for a non-trivial amount of time.[1]
3. We conclude that removal of a sensor node from the deployment area can be noticed by its neighbors, as well as by the sensor node itself. Using appropriate measures (see Section 6), the affected sensor node can be timely excluded from the network.
4. Therefore, we looked into possibilities to attack unattended sensor nodes *in the field*, without disruption of the regular node operation. We actually found some attacks (and also countermeasures) which are described in Section 5, and evaluated them in experimental setups.

Roadmap. First, in Section 2, we give an overview of previous work on physical and tampering attacks on embedded systems in general and how they relate to sensor networks. We also give pointers to existing work concerning forms of attacks against WSNs which we do not consider here. In Section 3, we describe current sensor node hardware. Section 4 gives our design space for attacking sensor nodes, and Section 5 presents some examples for attacks and defenses. In Section 6, recommendations on WSN design resistant against node capture attacks are given, and ongoing and future work is presented.

2 Related Work

Vogt et al. [33] consider the case of *recovering* nodes that have been the target of a successful attack and have fallen under the control of an attacker. They describe

[1] However, this applies only if the WSN designers take some basic precautions which are well known in the world of embedded systems, such as disabling the JTAG interface (see Section 5.1), or protecting the bootstrap loader password (see Section 5.2).

an intrusion detection and recovery mechanism which can be implemented in software on certain types of microcontrollers.

Bugs in the software running on the sensor nodes or on the base stations give rise to attacks which can be easily automated and can be mounted on a very large number of nodes in a very short amount of time. Possible countermeasures include a heterogenous network design and standard methods from software engineering [22, 11, 32, 16].

Physical attacks on embedded systems, that is, on microcontrollers and smart cards, has been intensively studied before [2, 26, 1, 25]. Skorobogatov describes in depth tampering attacks on microcontrollers, and classifies them in the three categories of *invasive*, *semi-invasive*, and *non-invasive* attacks [25]. Invasive attacks are those which require access to a chip's internals, and they typically need expensive equipment used in semiconductor manufacturing and testing, as well as a preparation of the chip before the attack can begin. Semi-invasive attacks require much cheaper equipment and less time than the invasive attacks, while non-invasive attacks are the easiest.

All of these attacks, including the so-called low-cost attacks, if applied to sensor nodes, would require that they be removed from the deployment area and taken to a laboratory. Even if in some cases, the laboratory could be moved into the deployment area in a vehicle, all attacks would require at least disruption of the regular node operation. Most of the invasive and many of the semi-invasive attacks also require disassembly or physical destruction of the sensor nodes.

Skorobogatov also lists several possible classification schemes, including U. S. government standards, both for attackers, according to their capabilities, and for defenses, according to their abilities to resist attacks from a certain adversary class.

The existing literature on physical attacks usually assumes that an attacker can gain unsupervised access to the system to be attacked for an extended period of time. This is a sensible assumption for systems such as pay-per-view TV cards, pre-paid electricity tokens, or GSM SIM cards. Attacks which may take days or even weeks to complete present a real threat to the security of these systems.

In wireless sensor networks, however, regular communication with neighboring nodes is usually part of normal network operation. Continuous absence of a node can therefore be considered an unusual condition that can be noticed by its neighbors. This makes time a very important factor in evaluating attacks against sensor nodes, as the system might be able to detect such attacks while they are in progress and respond to them in real-time. One of our aims has been to determine the exact amount of time needed to carry out various attacks. Based on these figures, the frequency with which neighbors should be checked can be adapted to the desired level of security and the anticipated threat model.

Finally, the focus of previous work has been mostly on attacking the components themselves as opposed to the entire products. Attacks on the circuit-board level have been deliberately excluded from many works, although they are recognized to be security-relevant in some cases. We did not exclude such attacks from our investigation since our focus was on the security of the entire node and not only of its individual components.

3 Current Sensor Node Hardware

Currently available sensor nodes typically consist of embedded hardware with low power consumption, and low computing power. A typical sensor node contains some sensors (light, temperature, acceleration etc.), a radio chipset for wireless communication, an EEPROM chip for logging sensor data, a node-to-host communication interface (typically a serial port), and a microcontroller which contains some amount of flash memory for program storage and RAM for program execution. Power is provided by batteries.

Figure 1 shows a general schematic for the hardware of current sensor nodes, while Figure 2 shows photographs of some concrete models available today.

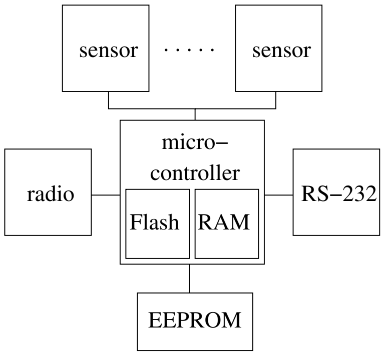

Fig. 1. General schematic of sensor node hardware

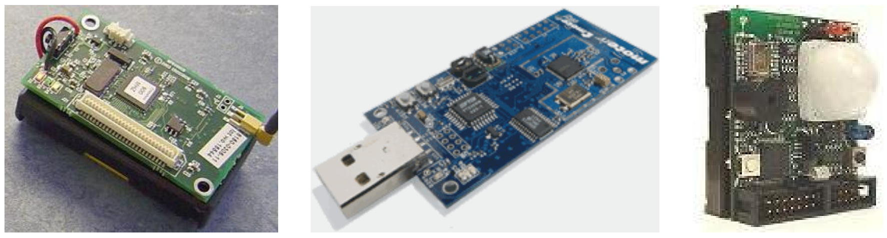

Fig. 2. Current sensor node hardware: Mica 2 by Crossbow, Berkeley (from [15]); Tmote sky by moteiv, Berkeley (from [20]); and Embedded Sensor Board by Scatter-Web, Berlin (from [5])

Typical choices for the microcontroller are the 8 bit Atmel ATmega 128 or the 16 bit Texas Instruments MSP430 family, with the amount of RAM varying between 2 kB and 10 kB and flash memory ranging from 48 kB to 128 kB. External EEPROM memory can be as small as 8 kB or as large as 1 MB. The speed of radio communications is in the order of 100 kbit/s.

The most interesting part for an attacker will no doubt be the microcontroller, as control over this component means complete control over the operation of the node. However, the other parts might be of interest as well in certain attack scenarios. Our classification scheme in Section 4 takes this into account, and Section 5 presents some examples for attacks on other components.

3.1 Mica2

The Crossbow Mica2 nodes [9,10] (fig. 2, left) use the 8 bit Atmel ATmega 128 microcontroller [4] with 4 kB RAM and 128 kB integrated flash memory, the Atmel AT45DB041B 4 Mbit flash memory chip [3], and the Chipcon CC1000 radio communications chipset [7] with a maximum data rate of 76.8 kbit/s. Programming is done via the Atmel's serial programming interface by placing the node in a special interface board and connecting it to an RS-232 serial port on the host.

3.2 Telos

The Telos motes [19,20] (fig. 2, center) by Moteiv utilize the Texas Instruments MSP430 F1611 microcontroller [31,30], providing 10 kB of RAM and 48 kB flash memory. The EEPROM used is the 8 Mbit ST Microelectronics M25P80 [27], and the radio chipset is the Chipcon CC2420 [8], whose maximum data rate is 250 kbit/s. Programming is performed by connecting to the USB interface and writing memory with the help of the MSP430 bootloader [28]. A JTAG interface is available as an alternative programming method and can also be used for debugging.

3.3 ESB

The Embedded Sensor Boards [5] (fig. 2, right) from ScatterWeb GmbH are built around the Texas Instruments MSP430 F149 microcontroller [31,29] featuring 2 kB of RAM and 60 kB flash memory, the Microchip 24LC64 [17] 64 kbit EEPROM, and the RFM TR1001 radio chipset [18] with a maximum data rate of 19.2 kbit/s. Programming is done either through a JTAG interface or over-the-air using a gateway.

4 Design Space for Physical Attacks on Sensor Nodes

4.1 Physical Attacks vs. Tampering

The majority of previous work on hardware security has focused on the security of single components. As different hardware platforms for wireless sensor networks are very similar to each other, we need not restrict ourselves to tampering attacks on the components themselves, but can include e. g. attacks on the circuit board level. This allows us to conduct a more detailed security analysis of the entire hardware instead of simply analyzing the security of every part on its own.

The term "tampering" is well accepted in the research community to designate attacks on components that involve modification of the internal structure of a single chip. At the same time, there are also conceivable attacks on sensor node hardware which do not really fit this accepted usage. In order to avoid this terminology problem, we call those attacks that we regard *physical attacks* and use this term to refer to all attacks requiring direct physical access to the sensor node.

4.2 Design Space

We propose the following classification scheme for physical attacks. This design space takes our previous considerations into account that sensor nodes are more or less in permanent contact with each other. This means that long interruptions of regular operation can be noticed and acted upon. Therefore, attacks which result in a long interruption (e. g. because the node has to be physically removed from the network and taken to a distant laboratory) are not as dangerous as those which can be carried out *in the field*. The main focus of our project has been on this class of attacks as well as on possible countermeasures to be employed by a sensor network.

The intention of our design space is to enable system designers to evaluate the feasibility and severity of a concrete attack under a concrete adversary model against their concrete system.

The two main categories that we use for classifying physical attacks are (1) the degree of control over the sensor node the attacker gains; and (2) the time span during which regular operation of a node is interrupted. Figure 3 illustrates this design space and classifies example attacks from Section 5 according to its criteria.

According to the degree of control the attacker gains, we classify the attacks into following categories, which are listed here in order of decreasing severity:

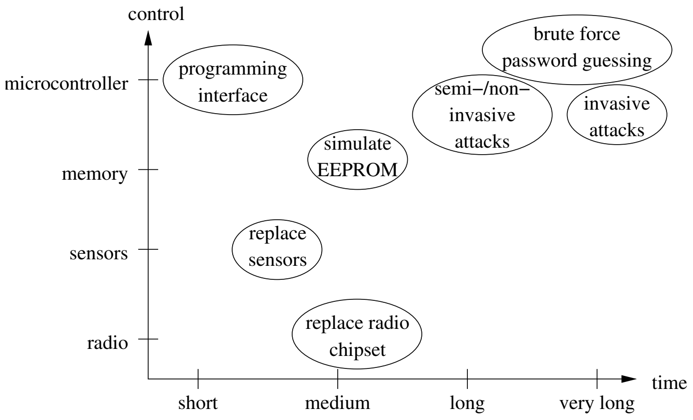

Fig. 3. Design space for physical attacks on sensor nodes

1. The attacker gains complete read/write access to the microcontroller. This gives the attacker the ability to analyze the program, learn secret key material, and change the program to his own needs.
2. The attacker learns at least some of the contents of the memory of the node, either the RAM on the microcontroller, its internal flash memory, or the external flash memory. This may give the attacker, e. g., cryptographic keys of the node.
3. The attacker is able to influence sensor readings.
4. The attacker can control the radio function of the node, including the ability to read, modify, delete, and create radio messages without, however, having access to the program or the memory of the sensor node.

We then classifiy the attacks according to the time during which the node cannot carry out its normal operation. In our design space, we use the following four possibilities:

1. Short attacks of less than five minutes. Attacks in this class mostly consist of creating plug-in connections and making a few data transfers over these.
2. Medium duration attacks of less than thirty minutes. Most attacks which take this amount of time require some mechanical work, for instance (de-) soldering.
3. Long attacks of less than a day. This might involve a non-invasive or semi-invasive attack on the microcontroller, e. g., a power glitch attack where the timing has to be exactly right to succeed, or erasing the security protection bits by UV light.
4. Very long attacks which take longer than a day. These are usually invasive attacks on the electronic components with associated high equipment cost.

Three other, less important from our point of view, properties associated with an attack can be illustrated as follows:

– The *cost* an attacker is required to spend in terms of equipment to carry out an attack successfully: This can range from extremely cheap, where only a soldering iron and some cables are required, to prohibitively high, where top-of-the-line semiconductor test equipment is needed.
– The *skill and knowledge* that an attacker has to possess for a successful attack: Some attacks might be carried out by a kid after proper instruction, while others might require extensive knowledge of the particular application of the sensor network, or a person trained in the use of special equipment. This property can also be modeled as *cost*.
– The *traces* left behind by the attack: If after the attack the node is left in the same state as before the attack, including unaltered memory contents, then this is harder to notice than an attack which causes physical destruction of the node.

5 Examples of In-the-Field Attacks and Countermeasures

Our project set out to actually implement attacks on wireless sensor networks. Our aim has been to gather some real-world data about currently known, proposed, and previously unidentified attacks. First, we wanted to measure the effort needed to carry out existing attacks, and the effort needed to devise new attacks. Second, we intended to evaluate the effectiveness of various existing security mechanisms and to come up with new ones. In the rest of this section, we will describe some of the attacks that we have investigated, together with possible countermeasures, and classify them according to the criteria described above.

5.1 Attacks Via JTAG

The IEEE 1149.1 JTAG standard is designed to assist electronics engineers in testing their equipment during the development phase. Among other things, it can be used in current equipment for on-chip debugging, including single-stepping through code, and for reading and writing memory.

A JTAG Test Access Port (TAP) is present on both the Atmel and Texas Instruments microcontrollers used on the sensor nodes described above. All sensor nodes examined by us have a JTAG connector on their circuit board allowing easy access to the microcontroller's TAP. While the capabilities offered by JTAG are convenient for the application developer, it is clear that an attacker must not be provided with the same possibilities. Therefore it is necessary to disable access to the microcontroller's internals via JTAG before fielding the finished product.

The MSP430 has a security fuse which can be irreversibly blown (as described in the data sheet) to disable the entire JTAG test circuitry in the microcontroller. Further access to the MSP430's memory is then only possible by using the Bootstrap Loader described in Section 5.2. The ATmega128 requires the programmer to set the appropriate fuses and lock bits, which effectively disable all memory access via JTAG or any other interface from the outside.

If JTAG access is left enabled, an attacker equipped with an appropriate adapter cable and a portable computer is capable of taking complete control over the sensor node. Even if there is no JTAG connector provided on the circuit board, attackers can still get access to the JTAG ports by directly connecting to the right pins on the microcontroller which can be looked up in the datasheet. Typical data rates for JTAG access are 1–2 kB/s, so reading or writing 64 kB of data takes between 30 and 70 s. However, there are specialized programming devices on the market which can attain much higher data rates. One such device claims to be able to program 60 kB of memory in a mere 3.4 s.

5.2 Attacks Via the Bootstrap Loader

On the Telos nodes, the canonical way of programming the microcontroller is by talking to the Texas Instruments specific bootstrap loader (BSL) through the USB interface. The bootstrap loader [28] is a piece of software contained in the ROM of the MSP430 series of microcontrollers that enables reading and writing the microcontroller's memory independently of both the JTAG access and the program currently stored on the microcontroller.

The BSL requires the user to transmit a password before carrying out any interesting operation. Without this password, the allowed operations are essentially “transmit password” and “mass erase”, i.e. erasing all memory on the microcontroller.

The BSL password has a size of $16 \cdot 16$ bit and consists of the flash memory content at addresses 0xFFE0 to 0xFFFF. This means in particular that, immediately after a mass erase operation, the password consists of 32 bytes containing the value 0xFF. The memory area used for the password is the same that is used for the interrupt vector table, i.e. the BSL password is actually identical to the interrupt vector table. The interrupt vector table, however, is usually determined by the compiler and not by the user, although Texas Instruments documents describe the password as user-settable.

Finding out the password may be quite time-consuming for an attacker. However, such an investment of time may be justified if a network of nodes all having an identical password is to be attacked. Therefore, an evaluation of the possibility to guess the password follows.

Brute Force. As the password is composed of interrupt vectors, certain restrictions apply to the values of the individual bytes. This section examines the expected size of the key space and estimates the expected duration of a brute force attack on the password.

Initially, the key space has a size of $16 \cdot 16\,\text{bit} = 256\,\text{bit}$. Assuming a typical compiler (mspgcc 3.2 [21] was tested), the following restrictions apply:

– All code addresses must be aligned on a 16 bit word boundary, so the least significant bit of every interrupt vector is 0. This leaves us with a key space of $16 \cdot 15\,\text{bit} = 240\,\text{bit}$.
– The reset vector, which is one of the interrupt vectors, is fixed and points to the start of the flash memory, reducing the key space to $15 \cdot 15\,\text{bit} = 225\,\text{bit}$.
– Interrupt vectors which are not used by the program are initialized to the same fixed address, containing simply the instruction `reti` (return from interrupt). As a worst case assumption, even the most basic program will still use at least four interrupts, and therefore have a key space of at least $4 \cdot 15\,\text{bit} = 60\,\text{bit}$.
– Code is placed by the compiler in a contiguous area of memory starting at the lowest flash memory address. Under the assumption that the program is very small and uses only $2\,\text{kB} = 2^{11}\,\text{B}$ of memory, we are left with a key space of a mere $4 \cdot 10\,\text{bit} = 40\,\text{bit}$.

We conclude that the size of the key space for every BSL password is at least 40 bit.

A possible brute force attack can be performed by connecting a computer to the serial port (the USB port, in the case of Telos nodes) and consecutively trying passwords. This can be done by executing a modified version of the `msp430-bsl` [21] program that is normally used for communicating with the BSL.

The rate at which passwords can be guessed was measured to be approximately 12 passwords per second when the serial port was set to 9600 baud. However, the MSP430 F1611 used on the Telos nodes is capable of a line speed of 38,400 baud, and at this speed, approximately 31 passwords can be tried per second. Finally, the BSL program normally waits for an acknowledgment from the microcontroller after each message sent over the serial line. If this wait is not performed, the speed of password guessing rises to 83 passwords per second.

The maximum speed of password guessing in practice can therefore be assumed to be 2^7 passwords per second. This is quite close to the theoretical limit of $38,400\,{}^{\text{bit}}/_{\text{s}} \cdot (256\,{}^{\text{bit}}/_{\text{pw}})^{-1} = 150\,{}^{\text{pw}}/_{\text{s}}$.

Recalling that the key space has a size of at least 40 bit, we can now conclude that a brute force attack can be expected to succeed on the average after $2^{40-7-1}\,\text{s} = 2^{32}\,\text{s} \approx 128\,\text{a}$. As 128 years is well beyond the expected life time of current sensor nodes, a brute force attack can be assumed to be impractical.

Knowledge of the Program. One consequence of the fact that the password is equal to the interrupt vector table is that anyone in possession of an object file of the program stored on a sensor node also possesses the password. Worse, even someone who only has the source code of the program still can get the password if he has the same compiler as the developer, since he can use this compiler to produce an image from the source code identical to the one on the deployed nodes.

The secret key in the current TinySec implementation, for example, is contained in the image but does not influence the interrupt vector table. If TinySec were ported to Telos motes, the source code and the compiler used would be sufficient information for an attacker to extract the secret key material. The same holds for any kind of cryptographic mechanism where the key material does not influence the interrupt vectors.

Another way of exploiting the identity of the password and the interrupt vector table is to take one node away from the network and attack the microcontroller on this node with classic invasive or semi-invasive methods. The absence of the node from the network will probably be noticed by the surrounding nodes and its key(s) will be revoked (see also Section 6). However, once the attacker succeeds with her long-term attack and learns the BSL password of the one node, it is trivial for her to attack all of the other nodes in the field if they all have the same BSL password.

If an attacker knows the BSL password, reading or writing the whole flash memory takes only about 25 s.

In order to avoid these forms of attack, we propose a technique called *interrupt vector randomization.* We have designed and implemented a program called `rand_int` that operates on a program image in Intel hex format produced by the compiler. It can be used to preprocess an image before installation on a node. Our tool reads the old interrupt vector table from the image file and replaces it in the following way:

1. While reading the image, all used memory areas are remembered and output unchanged.
2. When the interrupt vector table is read, it is not output directly.
3. Instead, for every original address in the interrupt table, an unconditional branch instruction to this address is generated.

4. Then an unused memory area is randomly chosen, the branch instruction is placed there, and that memory region is marked as used.
5. Finally, the entry in the interrupt table is replaced by the address where the branch instruction was placed.

The resulting image file then leads to a BSL password that can neither be guessed nor derived from the source code, effectively preventing third parties from reading the sensor node's memory even if they have access to the source code of the application used on the node, or if they have attacked a different node and learned its BSL password. Care should be taken to erase the image file with randomized interrupt vectors after programming the node.

This approach could also be extended for over-the-air reprogramming. Instead of performing the randomization process on the developer's host and sending an individual image to every node in the network, an identical image could be broadcast to all nodes. The randomization process would then have to take place on every individual node after it receives the new image.

5.3 Attacking the External Flash

Some applications might want to store valuable data on the external EEPROM. For example, the Deluge implementation of network reprogramming in TinyOS stores program images received over the radio there. If these images contain secret key material, an attacker might be interested in reading or writing the external memory.

Probably the simplest form of attack is eavesdropping on the conductor wires connecting the external memory chip to the microcontroller. Using a suitable logic analyzer makes it easy for the attacker to read all data that are being transferred to and from the external EEPROM while she is listening. If a method were found to make the microcontroller read the entire external memory, the attacker would learn all memory contents. This kind of attack could be going on unnoticed for extended periods of time, as it does not influence normal operation of the sensor node.

A more sophisticated attack would connect a second microcontroller to the I/O pins of the flash chip. If the attacker is lucky, the mote microcontroller will not access the data bus while the attack is in progress, and the attack will be completely unnoticed. If the attacker is skillful, she can sever the direct connection between the mote microcontroller and the flash chip, and then connect the two to her own microcontroller. The attacker could then simulate the external memory to the mote, making everything appear unsuspicious.

Of course, instead of using her own chip, the attacker could simply do a "mass erase" of the mote's microcontroller and put her own program on it to read the external memory contents. This operation is even possible without knowledge of the BSL password. While this causes "destruction" of the node from the network's point of view, in many scenarios this might not matter to the attacker.

The exact amount of time required for the attacks proposed above remains to be determined. It should be noted that some of the attacks outlined above require a high initial investment in terms of equipment and development effort. A possible countermeasure could be checking the presence of the external flash in regular intervals, putting a limit on the time the attacker is allowed to disconnect the microcontroller from the external flash.

5.4 Sensors

Sensor nodes rely on their sensors for information about the real world, so the ability to forge or suppress sensor data can be classified as an attack. For instance, a surveillance system might be tricked into thinking that the situation is normal while the attacker passes unnoticed through the area under surveillance.

Replacing sensors on the different types of nodes varies in difficulty between child's play and serious electrical engineering, mostly depending on the type of connection between the microcontroller circuit board and the sensors. A pluggable connection—as present on the Mica2 motes—requires an attacker to spend only a few moments of mechanical work. If, on the other hand, the sensors are integrated into the printed circuit board design, replacing them involves tampering with the conductor wires, cutting them, and soldering new connections. The amount of time required for this will vary with the skill of the attacker, but it can be assumed to be in the order of minutes.

5.5 Radio

Finally, the ability to control all radio communications of a node might be of interest to an attacker. At the moment, we do not have any concrete attack which involves replacing the radio chipset, but we believe that it can still prove useful in some attack.

6 Conclusion

We systematically investigated physical attacks on current sensor node hardware, paying special attention to attacks which can be executed directly in the deployment area, without interruption of the regular node operation. We found out that most serious attacks, which result in full control over a sensor node (*node capture*), require absence of a node in the network for a substantial amount of time. We also found simple countermeasures for some of the most serious attacks,

Thus, in order to design a WSN secure against node capture attacks, the following steps should be applied:

- take standard precautions for protecting microcontrollers from unauthorized access;
- choose a hardware platform appropriate for the desired security level, and keep up-to-date with new developments in embedded systems security;
- monitor sensor nodes for periods of long inactivity;
- allow for revocation of the authentication tokens of suspicious nodes.

Standard precautions for protecting microcontrollers from unauthorized access, such as disabling the JTAG interface, or protecting the bootstrap loader password, are an absolute prerequisite for a secure sensor network. We developed a method of protecting the bootstrap loader password by randomization of the interrupt vector table. This allows the developers to make source code of their products public without fearing that their WSN can be taken over by everybody who compiles the source code using the same compiler, thus obtaining the same interrupt vector table, and therefore, the same BSL password.

As security is a process, not a product, system designers should keep up-to-date with the developments in attacks on embedded systems. The security of important systems should be constantly re-evaluated to take new discoveries into account, as newly found attack methods on microcontrollers or previously unknown vulnerabilities might make a previously impossible low-cost attack in the field possible.

The level of security required from the application should also be kept in mind when choosing hardware. In some cases it might make sense to build additional protection, such as a secure housing, around a partially vulnerable microcontroller.

Finally, the removal of a sensor node from the deployment area can be noticed by its neighbors using, e. g., heartbeat messages or topology change notifications, as well as by the sensor node itself using, e. g., acceleration sensors. Appropriate measures can then be taken by the network as well as by the node itself. The network might consider a node that has been removed as "captured" and revoke its authorization tokens or initiate recovery when this node returns to the network. The node itself might react to a suspected physical attack by erasing all confidential material stored on it.

Mechanisms should be developed that allow a sensor node which has been absent for too long from the network to be revoked by its neighbors. This is our future work. Note that depending on the WSN design, local revocation could be insufficient. For example, if an attacker removes a single sensor node from the network and successfully extracts the node's cryptographic keys, the attacker would be able to *clone* nodes, to populate the network with new sensor nodes which all use the cryptographic keys of the captured sensor node. Thus, a WSN should also be protected from node cloning.

References

1. Ross J. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems.* John Wiley & Sons, Inc., 2001.
2. Ross J. Anderson and Markus G. Kuhn. Low cost attacks on tamper resistant devices. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 125–136, London, UK, 1998. Springer-Verlag.
3. Atmel Corp. AT45DB041B datasheet. Atmel document no. 3443, available at `http://www.atmel.com/dyn/resources/prod_documents/doc3443.pdf`.
4. Atmel Corp. ATmega128 datasheet. Atmel document no. 2467, available at `http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf`.

5. FU Berlin. ScatterWeb Embedded Sensor Board. Online at http://www.inf.fu-berlin.de/inst/ag-tech/scatterweb_net/esb/.
6. Haowen Chan, Adrian Perrig, and Dawn Song. Random key predistribution schemes for sensor networks. In *IEEE Symposium on Security and Privacy*, pages 197–213, May 2003.
7. Chipcon AS. CC1000 datasheet. Available at http://www.chipcon.com/files/CC1000_Data_Sheet_2_3.pdf.
8. Chipcon AS. CC2420 datasheet. Available at http://www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf.
9. Crossbow, Inc. MICA2 data sheet. Available at http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf.
10. Crossbow, Inc. MPR, MIB user's manual. Available at http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_Users_Manual.pdf.
11. Mark G. Graff and Kenneth R. van Wyk. *Secure Coding: Principles and Practices.* O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.
12. Joengmin Hwang and Yongdae Kim. Revisiting random key pre-distribution schemes for wireless sensor networks. In *SASN '04: Proceedings of the 2nd ACM workshop on Security of ad hoc and sensor networks*, pages 43–52. ACM Press, 2004.
13. Chris Karlof, Naveen Sastry, and David Wagner. TinySec: A link layer security architecture for wireless sensor networks. In *Second ACM Conference on Embedded Networked Sensor Systems (SensSys 2004)*, November 2004.
14. Chris Karlof and David Wagner. Secure routing in wireless sensor networks: Attacks and countermeasures. *Elsevier's Ad Hoc Network Journal, Special Issue on Sensor Network Applications and Protocols*, September 2003.
15. Geoff Martin. An evaluation of ad-hoc routing protocols for wireless sensor networks. Master's thesis, University of Newcastle upon Tyne, May 2004.
16. Gary McGraw and John Viega. *Building Secure Software: How to Avoid Security Problems the Right Way.* Addison-Wesley, September 2001.
17. Microchip Technology. 24AA64/24LC64 datasheet. Available at http://ww1.microchip.com/downloads/en/DeviceDoc/21189K.pdf.
18. RF Monolithics. TR1001 datasheet. Available at http://www.rfm.com/products/data/tr1001.pdf.
19. moteiv Corp. Telos revision B datasheet. Available at http://www.moteiv.com/products/docs/telos-revb-datasheet.pdf.
20. moteiv Corp. Tmote Sky datasheet. Available at http://www.moteiv.com/products/docs/tmote-sky-datasheet.pdf.
21. http://mspgcc.sourceforge.net/.
22. Holger Peine. Rules of thumb for secure software engineering. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 702–703, New York, NY, USA, 2005. ACM Press.
23. Adrian Perrig, John Stankovic, and David Wagner. Security in wireless sensor networks. *Commun. ACM*, 47(6):53–57, 2004.
24. Elaine Shi and Adrian Perrig. Designing secure sensor networks. *IEEE Wireless Communications*, 11(6), December 2004.
25. Sergei P. Skorobogatov. Semi-invasive attacks - a new approach to hardware security analysis. Technical report, University of Cambridge, Computer Laboratory, April 2005. Technical Report UCAM-CL-TR-630.
26. Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In *CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, pages 2–12, London, UK, 2003. Springer-Verlag.

27. STMicroelectronics. M25P80 datasheet. Available at `http://www.st.com/stonline/products/literature/ds/8495.pdf`.
28. Texas Instruments. Features of the MSP430 bootstrap loader (rev. B). TI Application note SLAA089B, available at `http://www-s.ti.com/sc/psheets/slaa089b/slaa089b.pdf`.
29. Texas Instruments. MSP430 F149 datasheet. Available at `http://www-s.ti.com/sc/ds/msp430f149.pdf`.
30. Texas Instruments. MSP430 F1611 datasheet. Available at `http://www-s.ti.com/sc/ds/msp430f1611.pdf`.
31. Texas Instruments. MSP430x1xx family: User's guide. TI Application note SLAU049E, available at `http://www-s.ti.com/sc/psheets/slau049e/slau049e.pdf`.
32. John Viega, Matt Messier, and Gene Spafford. *Secure Programming Cookbook for C and C++*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2003.
33. Harald Vogt, Matthias Ringwald, and Mario Strasser. Intrusion detection and failure recovery in sensor nodes. In *Tagungsband INFORMATIK 2005, Workshop Proceedings*, LNCS, Heidelberg, Germany, September 2005. Springer-Verlag.