Steel, Cast Iron and Concrete: Security Engineering for Real World Wireless Sensor Networks

Frank Stajano, Dan Cvrcek, and Matt Lewis

Computer Laboratory, University of Cambridge
15 JJ Thomson Av, CB3 0FD Cambridge, UK
frank.stajano@cl.cam.ac.uk, dc352@cl.cam.ac.uk,
ml400@cam.ac.uk

Abstract. What are the real security issues for a wireless sensor network (WSN) intended to monitor the structural health of a suspension bridge, a subway tunnel or a water distribution pipe? Could an attack on the sensor network cause the structure to collapse? How easy is it for civil engineers or other domain experts to build a secure WSN using commercially available hardware and software?
We answer these questions by conducting a qualitative risk assessment with bridge and subway tunnel operators and by conducting penetration testing on commonly available commercial WSN hardware and software, namely the Crossbow MICAz motes running TinyOS and XMesh and communicating over IEEE 802.15.4.

1 Introduction

We are interested in the practical security of real-world deployments of wireless sensor networks (WSN). The goal of our project is to develop reliable ways of monitoring the condition of large civil engineering structures such as bridges, subway tunnels, water pipes and sewers so that structural damage (due for example to corrosion, materials fatigue, overloading or shifting of surrounding soil) may be noticed and remedied before the structure weakens to the point of failure.

Traditionally, such monitoring is effected through periodic visual inspection. However the cost of accessing the structure for inspection is high, not just in terms of the effort required for the people in hard hats to reach the part of the structure to be monitored (think of the main steel cables of a suspension bridge or the middle section of a 3 km stretch of subway tunnel between two stations) but also in terms of the downtime of the structure for its regular users. This means that, unless there is reason to suspect a problem at a particular spot, routine inspections are infrequent (perhaps every few months) and cannot easily pick up new faults as they develop. This motivates our search for a solution based on a fixed network of sensors—wireless rather than wired in order to facilitate deployment in hard-to-access locations. Within the project, we (the three authors of this article) are responsible for the security engineering aspects of the wireless sensor network.

S.M. Bellovin et al. (Eds.): ACNS 2008, LNCS 5037, pp. 460–478, 2008.

From the security viewpoint, what is the problem to be solved? A preliminary challenge is to understand what damage could be caused by a failure (whether accidental or maliciously induced) of the sensor system. Would an attack on the sensor system just compromise the monitoring functionality, forcing the structure operators to fall back to manual inspection, or could it also have direct repercussions in the physical world, such as causing actual damage to the structure or to other nearby entities? If so, how? It is difficult to quantify such risks because there is essentially little or no prior experience of large civil engineering structures being monitored for many years by wireless sensors. Sometimes the structure operators themselves cannot imagine any serious security issues. It would be a mistake to launch into grand plans for encryption, key management and access control based just on what is fun for security researchers to do, rather than in response to recognised risks. So the consequences of security failures must be researched and understood, and the cost of security measures must be appropriate for the risks. The second challenge, assuming that the risk analysis requires and justifies such action, is to build a secure network of sensors using commercial off the shelf (COTS) hardware and software. Are current systems sold in a secure configuration? If not, is it still possible for non-experts to build a secure system out of such components? How? And how difficult is it?

In this paper we offer the following contributions. Firstly, we report on our initial qualitative risk assessment, carried out by interviewing the operating manager of a large suspension bridge and a contractor responsible for part of a large subway tunnel network (section 3). Secondly, and most significantly, we assess the practical security of the particular COTS system adopted by our team, the Crossbow MICAz motes running TinyOS or XMesh, together with the Stargate gateway: we design and implement a variety of attacks on this system and we report on the security problems we found, together with appropriate fixes where possible (section 4). As a further contribution to WSN security we ported the TinySec security library to the MICAz motes[1]. While some of our attacks exploit generally known vulnerabilities, others like selective jamming (section 4.3) and power exhaustion through routing table manipulation (section 4.5) are original and interesting in their own right. In section 4.3 we also demonstrate how an attacker can undetectably alter messages in an IEEE 802.15.4 radio environment. Finally, based on the experience we gained, we offer some architectural recommendations (section 5), independent of the particular hardware and software we used, that will help future teams design and deploy more secure WSN systems out of COTS hardware and software.

2 Scenario

2.1 Sensing

The purpose of a sensor network in our scenario is to monitor ambient conditions for hints that the structure may be deteriorating. Here is a non-exhaustive list of examples of what we monitor and why.

[1] The source code is available under GPL from `http://www.winesinfrastructure.org/`

Bridges. The main cables of a 2 km suspension bridge are almost 1 m in diameter; each of them is actually a bundle of over ten thousand 5 mm steel wires, all anchored in strands into huge concrete blocks in underground chambers at either end of the bridge. We monitor temperature and humidity at various points in those chambers for signs of conditions that might lead to corrosion.

Some of the wires do break over the lifetime of the bridge owing to corrosion, defects and stress. A large safety factor is built in by design, to ensure that the main cables will still support the weight of the loaded bridge despite a number of inner wires having snapped; but monitoring the rate of breakages is very hard. A visual inspection, involving opening up the main cables with wedges, is extremely costly and disruptive; it will only spot breakages that are close to the inspection point and it may itself cause further breakages by stressing the cable. We will instead be using sensors based on acoustic monitoring of the "ping" sound made by a wire snap: with several synchronised sensors on the main cables, one may approximately locate the position of a breakage. Such sensors require a 100 kHz sampling rate, which imposes stringent performance constraints on the nodes.

Tunnels. A subway tunnel is essentially a hollow underground burrow whose walls are lined with large cast iron or concrete tiles. As the surrounding soil moves, settles and subsides, the cross-section of the tunnel deforms and the walls of the tunnel get damaged, perhaps developing cracks. We monitor existing cracks with sensors that measure the displacement across the edges of the crack. We also use inclinometer sensors to measure whether a given tile moves (by as little as a hundredth of a degree), indicating deformation of the tunnel that might lead to further cracks. We also monitor relative humidity, temperature and vibrations.

Water pipes. A large water distribution pipe can be almost 1 m in diameter. The opening and closing of valves causes pressure waves that stress the pipe and may eventually result in leaks. A leak in a large water pipe may discharge substantial amounts of water and cause a local flood in less than an hour. Prompt intervention is essential. We monitor water pressure at various points in the pipe and infer the presence and location of leaks with mathematical models.

2.2 Network Architecture

The standard architecture for this type of application is a three-tier wireless sensor network. At the bottom tier, the sensors are attached to nodes[2] that form a multi-hop ad-hoc network. Modern motes tend to use IEEE 802.15.4 as their communication standard[3] but some older motes may use unlicensed frequencies such as 868 or 916 MHz.

At the middle tier, the data measured by the motes is routed to a gateway, physically located near the nodes (e.g. inside the tunnel) because of obvious

[2] Also known as *motes* from the seminal "Smart Dust" paper [1].

[3] This standard is often colloquially indicated as ZigBee but, strictly speaking, most current motes do not implement the higher layers specified by ZigBee on top of the PHY and MAC defined by 802.15.4.

connectivity requirements. The gateway, usually an embedded Linux PC, may perform some preprocessing on the collected data. Communication between the gateway and the next tier is frequently implemented via GPRS, which is easy to deploy in isolated areas. However, GPRS is relatively slow and expensive and therefore may not be suitable for applications that require high data rates or always-on connectivity (as opposed to, say, overnight batch logging of a few data samples collected during the day). It also does not work in tunnels. Where ADSL is available, it is usually more convenient.

At the top tier, the data collected by the gateway(s) is aggregated into a database running on a central server (usually a PC) that will be accessed by various applications for visualisation and processing of the data.

3 Qualitative Risk Assessment

Since the value of our project to the security community comes primarily from its link to real-world installations, we started by interviewing two senior representatives of organisations that respectively operate a large suspension bridge and a system of underground tunnels. We wanted to elicit their perception of risks rather than relying on our own guesswork. This is an ongoing portion of the project and we next expect to interview a water distribution operator.

Direct consequences in the physical world. The disruptive potential of data modification or injection attacks is greatly amplified if the sensors are directly connected to actuators in a feedback loop (e.g. actuators close the valves as soon as a leak is reported). In such cases, altering bits in a radio message has a direct effect on the real world (a mains water pipe is closed down, stopping water delivery for a whole neighbourhood).

Our first round of questions to the bridge and subway operators therefore aimed to elicit whether the WSN would plausibly be linked to any actuators once the system reached maturity. For the bridge, the answer was a definite no: none of the measurements would have consequences requiring automatic and immediate action. Everything is reported to a control room from where human operators have direct visibility of the bridge. Any action, including such "soft" actions as activating road signs that lower the speed limit for motorists (something that is done in poor weather conditions such as high winds, frost and so on), is initiated manually by an operator who cross-checks the sensor readings with other clues (e.g. a windsock). For the subway tunnels the situation was more subtle, since operators cannot rely on a visual cross-check, but our contact could still not imagine a situation in which tunnel sensors would be directly attached to actuators without human intermediation. We are keen to interview a water operator, though, whose answer might be quite different in light of the leak scenario.

Confidentiality. We also asked whether the operators would be worried if sensor readings were not kept confidential. The bridge operator was not worried at all, since from his experience he did not expect any such reading ever to say anything

extraordinary or embarrassing. The tunnel operator was more sensitive and cautious: if a problem occurs he wants to hear about it first, and have a chance to address it before others (e.g. the press) know it occurred. A well designed WSN system should therefore be flexible; encryption should only be activated if confidentiality protection is worth more than its cost in energy consumption, key management and system complexity.

Short-Term Integrity. Concerning the threat of false negatives (attacker making us believe that there is no fault when there is one, thus stopping the structure owner from carrying out maintenance that might fix the problem, leading to aggravation of the problem with time and possibly partial or total collapse) we learnt from these interviews that, over the life of these structures up to this point (26 years for the bridge; 118 years for the underground tunnels), the number of serious structural incidents has been nil or very low. This implies that trying to take down the structure by hiding spontaneous faults that the sensor network would report is a strategy that might force the attacker to wait for a very long time! The situation might be different if the attacker also *caused* the damage, as well as was subsequently trying to hide it from the sensors, but if this were perceived as a serious threat then safeguards against causing physical damage would be much more of a priority than those against network tampering.

With the injection of false positives, instead, the sensors report damage (e.g. several wire snaps one after the other) that has not actually occurred. This forces the maintenance operators to waste time trying to locate and repair a non-existent fault, e.g. by opening up the main cable with wedges without finding anything. It may also force temporary closure of the structure while the problem is investigated. Therefore, false positives would be more disruptive, since unlike false negatives they could be triggered at will by the attacker. In any case the feedback was that any major alerts from the new system (WSN) would be viewed with some suspicion by the operators until the system had proved its worth. This was particularly true for the bridge operator, for whom the new system was a nice extra but not a necessity, whereas the subway operator had no "eyes" in the tunnel and was therefore more eager to accept and trust any technological development giving him greater monitoring power.

Medium-Term Integrity. The subway tunnel operators see value in analysing sensor data over the medium term (e.g. months) because they currently have no systems allowing remote monitoring of the state of the tunnels; and the tunnels themselves can only be physically inspected in the middle of the night when trains do not run. Systematic collection of data about the state of the tunnels would be very useful in budgeting for maintenance costs as it would allow more precise estimates of necessary works. Maintenance budget negotiations are currently based on very vague estimates derived from manual inspection of a small sample of accessible sections of the tunnels. Continuous monitoring would provide more reliable quantitative estimates that all parties involved would have an easier time accepting.

Long-Term Integrity. For structure operators, a significant perceived benefit of our automated sensing is the provision of decades-long monitoring logs that will

allow them to carry out previously impossible research on long term behaviour (and, inevitably, decay) of materials when plotted against influencing factors such as humidity, acidity, pollutants and stresses. The monitoring thus becomes particularly valuable when it allows reliable data collection over many years. One requirement is therefore that data be collected and stored in a sensibly designed non-proprietary format that can still be read after decades, under the assumption that any component of the physical implementation of the system will be replaced by something newer in due course. Another consequent requirement is on data integrity: corruption of sensor readings, especially if on a small enough scale as to go undetected at acquisition time, would invalidate the whole historical database and make the exercise useless. This requires an architecture in which, regardless of confidentiality, integrity of sensor readings is preserved at all costs.

Availability. We explained that it would be technically impossible to eliminate denial of service attacks on a wireless network and tried to understand their practical consequences for the operators. Apart from the financial loss of having wasted money on an unusable sensor system, even a complete jamming of the wireless network did not appear as a grave loss to the bridge operator, who did not expect ever to depend entirely on the WSN output; but it sounded somewhat more embarrassing for the subway operator, who anticipated his newly gained real-time "eyes" on potential cracks as a facility he would not want to do without.

Conclusions. The general conclusion from the interviews with the structure operators is that data integrity is the most important security property for this type of application. No direct link from sensors to actuators is envisaged in the two systems we discussed, so the main effects of an attack on the WSN are invalidation of collected data or incapacitation of the WSN system rather than damage to real world facilities. This is therefore not a very high risk setting. Having said that, there is still scope for attackers causing disruption to users of the structure insofar as denial of service or injection of false positives may lead the operators to close the bridge or the tunnel for safety reasons while the problem is investigated.

4 Attacks on a Real System

Inspired by the above-mentioned user concerns, we examined the available WSN technologies for vulnerabilities threatening the desired security properties. As we did that, we also found problems that the operators had not anticipated.

Our goal was to assess practical security of wireless sensor networks on a real, physical system, as opposed to just in theory or through simulations. So we targeted our attacks on the particular platform that our project adopted, namely the MICAz mote from Crossbow, running TinyOS v1.1 and XMesh from MoteWorks 2.0.F, together with the Stargate rev. 1.2 as a gateway.

A superficial reader might comment that, since we chose the components and assembled the system ourselves, any security holes we find only reflect on our own incompetence. On the contrary, the spirit of our investigation was to

imagine that a team of application experts (in this case civil engineers), assumed to be security-conscious but not security experts, puts together a system using COTS components, following the manufacturer's instructions and activating any recommended security features. We set out to assess the practical security of the resulting system and to suggest ways of improving it where appropriate.

Our limited budget and manpower would never have allowed us to carry out a comparative study of all commercially available WSN platforms to determine the most secure one, so that was never a goal. Nonetheless, we believe our results will interesting for users of other platforms too.

Each of the attacks or exploits described in this section has been carried out and validated on actual hardware. We report sufficient details to convince the reader that a vulnerability exists and has been exploited by us, but stop short of supplying malicious readers with a cookbook. We also describe how to fix the problem wherever possible. As a courtesy we supplied a copy of a preliminary version of this paper to Crossbow in September 2007, to give them a chance to release security patches based on our advisories.

4.1 Our Platform

The TinyOS operating system runs on various hardware platforms including MICA2, MICAz, Iris (Crossbow Inc.), Tmote (Moteiv), Intel Mote, Intel Mote 2 (Intel). It is a modular system that allows easy extension with drivers for new sensor boards or functionality.

TinyOS v1.1 appears to be the most commonly used version in practice: v2 exists but is not stable yet. TinyOS may be deployed as is, or in conjunction with a commercial derivative such as XMesh from Crossbow or Boomerang from Moteiv. Being an open source project, TinyOS is reasonably easy to analyse for stability and security issues. TinyOS v1.1 has been stable since September 2003, so it can be considered a fairly mature product.

We began by analysing TinyOS, focusing our attention primarily on cryptography and routing protocols. The first big surprise was that, while TinyOS ships with the cryptographic module TinySec [2], the latter can only be compiled for MICA2 motes and there is no implementation available for the current generation of motes with 802.15.4-compliant radio chips. This means that all the networks based on modern Crossbow, Moteiv or iMote devices are vulnerable to a slew of attacks from even relatively unskilled attackers. To address this deficit, we ported TinySec[4] to make it run with the latest 802.15.4 chips—namely the Texas Instruments CC2420 chip, used in most of the motes mentioned above—so that we could test our attacks on cryptographically secured networks. In the process of performing this port, we noticed that the TinySec MAC generation code uses a fixed block size of 7 bytes, while the underlying block cipher has a fixed block size of 8 bytes. While we do not believe that this causes a security exposure, our port corrects this behaviour to use the same block size

[4] See footnote 1.

as the underlying cipher[5]. Note that, as the TinySec authors themselves point out, TinySec provides no protection against replay attacks. Unfortunately this protection does not exist at any layer of the TinyOS radio stack either. In addition, since by design TinySec only supports a single key shared across the entire network, there is no protection against physical capture of one mote.

A general virtue (security-wise) of the TinyOS code is that it is very lean and spartan. The underlying system supplies minimal functionality, preferring to export simple primitives to applications. One of the results of this is that the basic TinyOS system seems to be fairly secure without making much apparent effort to be so; several times during the course of our analysis we tried out attacks that we thought might work, but were blocked by TinyOS' minimalist philosophy. In contrast to this, the commercial system we analysed, Crossbow XMesh, exports a very rich set of features to applications running on it and consequently exposes a much wider attack surface.

Concerning power consumption, a mote using its radio continuously would exhaust a pair of alkaline AA batteries in a couple of days (a couple of weeks for the expensive D lithium batteries we use). In normal use, with a duty cycle of around 1%, a mote lasts for several months.

4.2 Classes of Attacks

We chose not to concentrate on **physical attacks** [3] on the sensors and on the nodes attached to them, not because we think they are impossible[6] but because an attacker with physical access to the sensors could with comparable effort stage much more destructive attacks on the structure itself, for example by using explosives. We therefore focus on attacks on the communication systems, primarily the ad-hoc radio used by the sensor nodes but also the back-end link from gateway to central server.

We studied three broad types of attacks: **data payload attacks** that change the content of data packets; **network attacks** that affect the functionality of the network, for example by preventing communication, taking down specific links, modifying the routing topology or rewriting the firmware of a node; and **system attacks**, potentially the most damaging, in which the attacker exploits a vulnerability in one part of the system architecture (e.g. the wireless network) to gain control of other parts (e.g. the gateway or the central computer).

Attack mechanisms we employed included jamming (at various degrees of selectivity and at different layers in the stack), replay attacks, packet injection or corruption (where the injected or malformed packets were specifically crafted to probe for vulnerabilities or to trigger known vulnerabilities) and ACK spoofing.

[5] This divergence from a correct implementation will become a compatibility issue if end-to-end keys are used, as the MAC algorithm will have to be implemented on both the motes and the gateway. The NesC code for the motes cannot be compiled for the gateway, so its use of non-standard parameters hinders interoperability.

[6] On the contrary, with so many unsupervised nodes over a large area, we believe one cannot exclude that a few nodes may at some point be physically captured, even though attackers are unlikely ever to be able to take over a *majority* of the nodes.

4.3 Jamming

While it is well known that no one can prevent physical jamming of the radio channel, appropriate design decisions can reduce a system's vulnerability to denial of service (DoS) attacks and increase the cost of such attacks. In this section, we investigate the baseline case of an attacker using an unmodified COTS mote as the transmitter. It is understood that a more resourceful attacker, with a higher power transmitter, could cause greater damage.

Description. We used a MICAz mote to jam the communication between other motes by transmitting at the same time as them, thereby causing a collision. Unlike most previous jamming attacks, ours is selective and jams only a subset of the packets (e.g. those coming from a specified victim) while leaving others unaffected. The attack can therefore be used to selectively disconnect individual motes or whole regions from the network. It is also hard to detect because unaffected nodes do not notice anything unusual.

The algorithm is as follows:

1. We select criteria for messages to be jammed, e.g. senderID = 3 and messageType = `AM_MULTIHOP`.
2. We compile the code with the selected criteria into the attacking mote and deploy the mote in the vicinity of the victim.
3. When the attacking mote detects a transmission, it listens to just enough of the message to determine whether the packet meets its jamming criteria.
4. If the message meets the criteria, go to step 5. In our example, we have a match if byte 13 = 3 and byte 9 = `AM_MULTIHOP`. Loop back to step 3 if the criteria were not met.
5. The attacking mote switches the radio chip to transmit mode and jams the rest of the message by transmitting on the same channel.

Steps 3 and 5 are difficult for technical reasons. To start with, the CC2420 is a packet based radio, so in normal operating conditions the microcontroller is only informed of radio activity after a complete packet has arrived. This obviously makes step 3 difficult since, by the time we can tell that we should jam a packet, it has already been transmitted. We were able to overcome this difficulty by putting the radio chip into a debugging mode, where each bit received from the radio was sent to one of the microcontroller's input pins as it arrived.

Our second problem in implementing the attack arose from the tight timing requirements imposed by the data rate of the radio—250 kbps. The ATMega128 microcontroller is clocked at 7.37 MHz, effectively giving us 30 instructions to process each bit. This would not be too much of a problem were it not for the fact that invoking a hardware interrupt appears to take around 30 clock cycles, meaning that our interrupt handler (which gets invoked once per bit) was always too slow. However, having the first invocation of the interrupt handler busy-wait for the remaining bits in the frame allowed us to meet the timing requirements.

Finally, switching from receive mode to transmit mode takes some time, so if the frame was particularly short we would frequently miss the end of the packet by the time we started jamming. By iteratively optimising the code, we were able to reliably jam frames with as few as 5 bytes of data.

Empirical Results. The 802.15.4 standard implemented by the motes uses direct-sequence spread spectrum (DSSS) to increase resilience to noise and jamming. However, since 802.15.4 specifies the spreading code to use, our motes shared the spreading code with the target, thereby nullifying the protection offered by DSSS. We carried out several experiments to measure the success rate of jamming. The experiments used three motes: transmitter, receiver, and jammer. The transmitter sent out a packet every 200 ms and the receiver, connected to a laptop, forwarded the received messages to a laptop where they were logged. All three motes were positioned 45 cm above the ground.

The jamming appeared successful in an open space, but with some anomalies. All messages were jammed when the attacking mote was between the sender and the receiver. The initial results (see Figure 1) suggested that the angle defined by transmitter-receiver-jammer had a significant impact on the success of the jamming, but later experiments showed a lack of repeatability, with variations between 30 and 88% for a given position. Complete jamming (100% of frames jammed) was achieved in several configurations of the three motes.

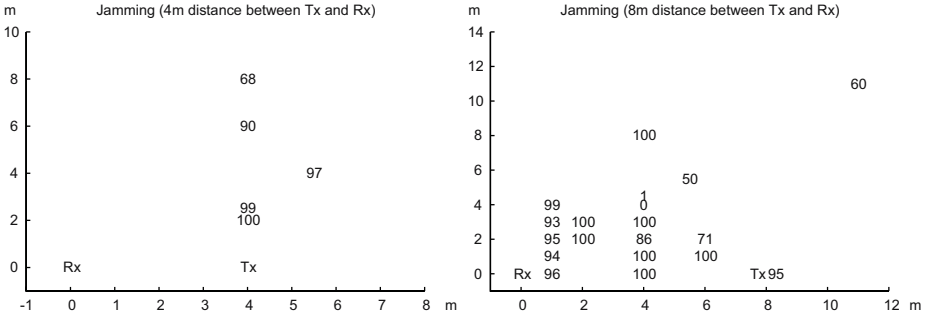

Fig. 1. Success rate of jamming depending on the position of the attacking mote. The Tx and Rx labels are the transmitting and receiving motes. The numbers 0–100 in the graphs denote the percentage of packets that were jammed when the attacking mote was in that 2D position relative to Tx and Rx (section 4.3).

We tried to find out why our results were so unpredictable, especially when the motes were close to the ground. A radio engineer told us that this is due to interference between the direct ray and the ray reflected from the ground, as the path difference between the two rays can be close to a half wavelength when the motes are on the ground. This results in destructive interference between the two rays, causing a weak signal. We think it may be some kind of voodoo.

Risks. The ability to selectively jam frames allows an attacker to:

- Make specific sensors deaf and/or mute;
- Prevent any sensor data from reaching the database, by jamming close to the gateway;
- Perform a man-in-the-middle attack by having a second mote listen for jammed frames and resend chosen ones with altered data.

The first two of these are simple DoS attacks and so may be written off by some as unimportant. The ability to alter the contents of any message while remaining undetected is rather more serious, as it gives the attacker complete control over the output of the network and therefore indirect control over all the systems controlled by the data reported by the sensor network.

Since the jammed frames are silently dropped by the radio chip, TinyOS is never notified. As such, a mote is unable to determine that a jamming attack is taking place unless its radio driver has been specifically modified to run in debug mode in order to detect this.

Underlying Cause. The vulnerability of a network to jamming as described here is simply a result of communicating over a shared medium to which adversaries have access (in this case radio).

Fix. To make it more difficult for an adversary to perform this type of selective jamming attack, the frame header could be moved to the end of the frame. This would mean that by the time an attacking node had decided to jam a frame, it would be too late. Unfortunately this defence is not perfect and succumbs to an attacker with two motes: one to jam every frame and one to resend any frames that do not meet the jamming criteria[7]. It is also expensive in terms of battery life, as each node must listen to and buffer the whole packet, instead of just the header, before knowing whether it was destined for itself. Detection of this type of jamming could be achieved by having one or more "observer" motes running in debug mode and listening for suspicious patterns of corrupted frames.

4.4 Counter Overflow Attack

Description. Another attack that can be mounted against a wireless network is a replay attack. This type of attack works even against an authenticated and encrypted network. One of the simplest protections against it is to use monotonically increasing counters to ensure the freshness of messages. TinyOS and TinySec use a 16 bit counter to distinguish messages, with TinySec using another counter as part of the encryption initialisation vector. Although TinySec explicitly does not offer replay protection, even if it did, the attack presented in this subsection would still work, unless TinySec increased the length of the counter. The network stack decides whether to accept a received message as valid according to the value of the counter in the message[8]. If the counter in the message is higher than the stored counter, the message is accepted. If it is lower but the difference is within an allowed range, the message is also accepted. Any

[7] We are not discussing all the details related to the necessity for the attacker to learn the whole content of every packet it jams in order to retransmit it if needed. This can be achieved in several ways, including jamming a part of the message that the attacker can reconstruct (such as the CRC, provided that there were no real errors), or jamming the message twice in different places, counting on the fact that the sending node will retransmit—although this latter strategy would be less stealthy.

[8] This is also used to compute the number of missed messages and subsequently the quality of the link with that neighbour.

other value causes the message to be rejected and the stored counter value to be zeroed. Each mote tracks the counter values of all its neighbouring motes.

For a replay attack to work, the receiving node needs to be ready to accept a message with the same counter value as the one we eavesdropped and intend to replay. We can achieve this by having the recipient's counter overflow and wrap around to the desired value. We thought of at least three methods to cause a counter overflow. First, we can inject fake messages that the target mote will forward, causing the target's counter to increment once per message we send. Second, we can use the routing table attacks described in section 4.5 to create a loop in the network: this causes a packet storm which eats up counter values. Finally, we can simply jam message acknowledgements from the target's parent as this will cause the target to retransmit each message several times.

Note that, if XMesh receives a message whose counter is set to zero, it bypasses the routing mechanisms used to verify the freshness of the message, obviating the need to overflow the counter. In addition, the receiving mote will not increment its stored counter, which means that the message has no effect on the link quality calculations. An attacker might exploit such "features".

Risks. When the message counter overflows we may initiate a replay attack with previously eavesdropped messages, even if these messages feature cryptographic integrity protection. This allows us to inject false negatives, false positives, or just slightly corrupted data to invalidate the long term monitoring. It also allows us to manipulate routing messages, even if they are authenticated. The message counters form the basis of many cryptographic mechanisms that rely on freshness, particularly in the absence of timestamps. This vulnerability would affect security mechanisms built from these cryptographic primitives.

Underlying Cause. Our three counter overflow methods rely respectively on the lack of source authentication, on the possibility of routing table manipulation (section 4.5), and on jamming (section 4.3). As such, the underlying causes of the counter overflow attack are the union of the underlying causes for these sub-attacks.

4.5 Routing Table Manipulation

Description. Nodes in an XMesh network populate their routing tables by listening to periodic broadcasts issued by their neighbours, which contain:

- The node's current parent;
- The path cost of sending a message from the node to the base station;
- A list of some of the node's neighbours and an estimate of how well it can receive messages from each.

When a node receives such a message from a neighbour, it updates its internal table of neighbours, which it consults every few minutes[9] to choose a new parent. All future messages are routed through this parent. The new parent appears to

[9] Every 30 seconds if the node is disconnected.

be chosen to minimise the path cost of sending messages back to the base station, with the link quality to each potential parent being a factor in the cost of the hop to the parent. The routing algorithm may thus be thought of as a distributed single-source shortest path algorithm.

Since all communication in the XMesh network is unauthenticated and unencrypted, an attacker can interfere with routing messages and thereby change the topology of the network to suit her whims. We created a Python module called `wsn` to read, modify, create and inject messages into the sensor network and we used it to implement and validate these routing attacks.

Risks. The ability to choose a topology for the network gives the attacker a lot of power. For example, she can cause all the traffic in the network to be routed through a mote she controls, allowing her to selectively drop or modify any message. Alternatively, she can cause a direct attack against the motes themselves by creating routing loops where, e.g. one route is $2 \mapsto 3 \mapsto 2 \ldots$ In such a configuration, nodes 2 and 3 will send messages back and forth, rapidly consuming their batteries. Using this method, we have been able to increase the peak rate at which nodes send messages to over 300/s with a mean value of around 25 messages per second. This can be used for a very powerful "sleep deprivation torture" [4] attack.

There is a low power implementation of XMesh that puts radio and microcontroller to sleep and they wake up only when necessary. However, any mote forwarding for at least one of its neighbours must wake up the radio chip regularly. The XMesh low power mode wakes up the radio chip roughly every 125 ms and it listens for a period of 1 ms. This means that the duty cycle is around 0.8%. The attack increases this value substantially to tens of percents.

Karlof and Wagner [5] discuss other routing attacks.

Underlying Cause. The problem here seems to be that the topology of the network is decided by the motes themselves while they do not have enough trusted information about the physical structure of the whole network to make sensible decisions. The ad hoc nature of the network appears to cause significant security issues, as everything received is trusted.

Fix. A natural solution to such trust issues would be to cryptographically authenticate messages sent by legitimate nodes. We could, for example, have all the nodes in the network share a key and use TinySec to authenticate and encrypt all messages sent over the network. This would mean that an attacker would not be able to forge routing messages without knowledge of the key. One problem with this scheme is that physical capture of a single node would reveal the network key and render the entire network susceptible to attack. Another problem is that authentic routing messages from one part of the network may still be recorded and replayed in other parts of the network, causing routing anomalies. We could stop this type of attack by using per-link keys (unique keys shared by each pair of neighbours), at the cost of complicating the key management process.

No cryptographic fix, however, would stop what has been called a "flagpole" attack, in which the attacker moves a victim mote up and down a flagpole in order to make other motes waste their battery updating their routing tables.

4.6 Over-the-air Programming

Description. One of the optional features that can be compiled into the motes making up an XMesh network is over-the-air programming (OTAP). This is a mechanism by which an entire network of motes can be reprogrammed remotely by sending them the code to execute. If OTAP is enabled on a network, no authentication is required to request a reprogram; an attacker able to send traffic to the network can therefore cause every mote to execute code of her choice.

Risk. The ability to reprogram motes allows an attacker to entirely control all data sent by the network. This means that she would be able to choose whether to send data from the sensors, what the apparent readings of the sensors were and how often the readings were reported.

Underlying Cause. The underlying problem here seems to be that there is no concept of authentication between any parties communicating over the network. As a result, all traffic is trusted and this is clearly a bad transport over which to run critical protocols such as OTAP.

Fix. The obvious action to mitigate this risk would be not to use OTAP. One fix could be to authenticate OTAP messages, e.g. with a scaled down version of the CMS standard for secure firmware update (RFC 4108). Using public key cryptography, generally thought of as computationally infeasible on low-end hardware, might be acceptable for such rare events as authenticating the signature of an OTAP request. Then the network would not be vulnerable to the physical capture of one node and so OTAP could be used securely.

4.7 Remote Command Execution in XServe

Description. XServe is a middleware component that connects the WSN to the back-end. If XServe is started with the `-h` flag, it starts a web server on a user-specified port. This web server is intended to be used to display the output of the attached sensor network. Unfortunately, one of the scripts supplied with the web server contains a bug that can be exploited by an attacker to execute arbitrary commands on the computer running XServe.

Risk. XServe may be run on a Stargate gateway or on the top-tier server. If the web server were enabled on a Stargate and an attacker could access the web server port, she could gain shell level access to the Stargate, obtaining complete control (read, write, modify, drop) over all the data sent from the sensor network to the central servers. If the web server were running on a central database server, an attacker could gain access to the administrative network, potentially gaining complete control over all new data from the network, as well as the ability to modify historical data and attack other parts of the network.

Underlying Cause. This exposure results from a simple programming error, but the underlying problem is really that the XServe component is far too feature rich. From a security viewpoint it is hard to defend the decision of equipping it with a built-in web server, for example.

Fix. We can fix this vulnerability by patching the script or by not using XServe's built in web server. However, since software errors cannot be eliminated, we suggest running complex software on safely stored data, outside the critical parts of the system.

4.8 Stargate Unhardened to IP Attacks

Description. The Stargate we bought new in 2007 shipped with very old versions of several pieces of software, many of which contained exploitable vulnerabilities. Some of the more serious issues are summarised in Figure 2.

Component	Exposure	CVE Reference
OpenSSH	Local root	CVE-2002-0083
		CVE-2003-0682
OpenSSH	Remote root	CVE-2003-0693
		CVE-2003-0695
Kernel	Remote root	CVE-2004-1137
Kernel	Local root	CVE-2005-1263
Kernel	Local root	CVE-2004-1235
PostgreSQL	Remote shell	CVE-2005-0247
PostgreSQL	Remote shell	CVE-2005-0245
PostgreSQL	Remote shell	CVE-2003-0901

Fig. 2. Some vulnerabilities affecting the Stargate. CVE reference numbers may be resolved at `http://cve.mitre.org/cve/`. (Section 4.8).

In addition to these problems caused by outdated software, the Stargate ships in an insecure configuration: it has a default, weak root password and an SSH daemon installed. In addition, if the PostgreSQL database is installed on the Stargate (a recommended and supported configuration), a database super-user is added with a default, weak password. The documentation does not explain how to change these passwords, nor does it suggest that you do. Furthermore the existence of the database super-user account is not even mentioned.

Risks. A remote attacker with access to the Stargate's IP interface may be able to gain root access or crash the Stargate. This would result in either a complete compromise of the sensor network or a complete DoS of the network.

Underlying Cause. The Stargate was not designed for security, as demonstrated by the outdated software, weak passwords, lack of security related documentation and unnecessarily high number of services running.

Fix. Patch, configure, minimise. If possible, update the software on the Stargate, in particular OpenSSH, PostgreSQL and the kernel. Do not run `sshd` on the Stargate and drop all inbound IP traffic to the Stargate that is not already part of an established TCP stream. This implies that all connections to the back-end network would have to be initiated by the gateway. In addition, change passwords for the system root account and the `tele` user in the PostgreSQL system.

5 Architectural Recommendations

Securing the Gateway. The WSN gateway is a bottleneck through which all data related to the sensor network flows. It is therefore crucial that it be properly secured. The usual configuration has the gateway performing many complex operations on the received data and exporting several services to the IP network. We instead recommend not to offer any services to the IP network (e.g. no SSH or HTTP daemons) and not to perform any analysis of the sensor data on the gateway. The gateway should act only as an SSH *client*, rather than server, and simply relay data from the WSN to the server. This would significantly reduce the attack surface of the gateway, resulting in greater security.

Key Management. Clearly, TinySec's default key management approach with a single key for all devices is inadequate to protect an unattended WSN, as the capture of a single node would expose the entire network. The use of pairwise link keys between motes would not protect against attacks on the gateway and another cryptographic protection would have to be used for communication with the central server. Based on the risk assessment in section 3, we recommend the use of pairwise end-to-end keys between each node and the central server to protect the integrity of the sensor data and to provide source authenticity. Insofar as routing is crucial to the security of the network, based on the vulnerabilities exposed in section 4.5 we also recommend the use of pairwise end-to-end keys between each node and the gateway to protect routing table data.

Routing. Many papers on WSNs axiomatically accept the smart-dust-derived assumption that individual nodes (possibly dropped from an aircraft) know nothing about each other's position and that therefore the network can only be built in an ad hoc, decentralised fashion. But, in our scenario, the deployment engineers *know* where they want to place the nodes—for example where they see cracks that need monitoring. We therefore suggest taking advantage of that positional information, perhaps by precomputing an initial (though sub-optimal) set of routing tables and preloading it into the nodes. We also suggest that routing calculations be performed centrally, for example at the gateway, after collecting authenticated local connectivity information from the nodes, since visibility of the whole connectivity graph would allow for the discovery of a better global solution. We are currently working on such a system.

Reviewing the risk assessment. Since the monitoring operation might last for several years, the risk assessment should be repeated at regular intervals to ensure that it still reflects the current usage patterns. It is common for systems to be used in ways that were not originally envisaged, so the protection goals and consequent security measures should be kept up to date.

6 Related Work

Although very many papers have been written about wireless sensor networks, experience papers reporting on real-world deployments are a minority: they include at least Mainwaring et al [6] who monitor seabird nesting environment and

behaviour, Arora et al [7] who deploy a perimeter control WSN and Werner-Allen et al [8] who monitor an active volcano. Closer to our scenario are Krishnamurthy et al [9] who monitor equipment for early signs of failure and especially Kim et al [10] who monitor the structural health of the Golden Gate bridge.

Similarly, although there is a vast literature on security of WSN, including such milestones as Perrig et al [11] on efficient broadcast stream authentication, Eschenauer and Gligor [12] on random key predistribution, Hu et al [13] on secure routing and Chen et al [14] on energy efficient topology maintenance, few such papers deal with attacks on actual sensor network platforms. One notable exception is the brilliant work by Becher et al [3] on physical attacks, providing concrete data on the effort required to extract secrets from a mote. Closer to our own investigations on 802.15.4 jamming are Wood et al [15], who discuss jamming attacks and countermeasures in detail, although their focus is on energy efficiency for the attacker rather than on being able to target a specific victim selectively. The TinySec work of Karlof et al [2] is of particular significance since it resulted in running code which we used extensively. One promising effort in a similar vein is that of Luk et al [16] for which, however, the code was only released after we completed this paper.

Our specific interest lies at the intersection of these two sets: real-world WSN deployments and real-world WSN attacks; so far we have not been able to find other papers in this subset. We believe that practical security is a field worth exploring in greater detail before proceeding with actual deployments.

7 Conclusions

What are the risks of a WSN that monitors large engineering structures? In the situations we examined, the sensors are never linked to actuators; therefore, deploying a WSN in such circumstances does not introduce major new risks. Typically, the worst outcome of an attack is that data gathered from the WSN will be useless, not that the structure itself will be directly endangered. Of course, if the WSN or the collected data becomes useless, there is a financial loss; moreover, false alarms might cause secondary losses through downtime; so, ensuring the integrity of the WSN is still a worthy goal. Considering our qualitative risk assessment (section 3), all the attacks presented in section 4 are potentially relevant, because they can be used directly or indirectly to corrupt integrity of sensor data—the main concern of the structure owners as expressed during interviews. The implementation of countermeasures, which must crucially protect the whole system including back-end and gateway as opposed to just the motes, could be prioritised based on a quantitative risk assessment that established the likelihood of each attack.

Will a WSN built by well-intentioned and security-minded application experts be secure by default? Based on the components we examined, no. For a start, if users of MICAz motes (or any other motes using 802.15.4) wanted to "turn on the crypto", they would find that TinySec does not even compile for their platform. Porting the code is not a trivial endeavour; fortunately, we have

now done it for them. Besides that, we have found and exploited a variety of vulnerabilities. The most devastating in practice are probably the most trivial from an intellectual viewpoint: the Over-The-Air Programming vulnerability (section 4.6), which allows an attacker to reprogram motes at will, and the Remote Command Execution vulnerability (section 4.7), which allows an attacker to gain complete control of the gateway or even the central server. Until such features are patched or protected, it is advisable to keep them disabled.

Among our other attacks, the most significant for the security researcher is probably our sleep deprivation torture attack based on routing table manipulation (section 4.5): it is crippling and very efficient because the attacker can set it up and walk away, as opposed to having to keep talking to the victims to drain their batteries out. Our selective jamming (section 4.3), too, is of interest as it is undetectable by OS and applications and can be used as the basis for more sophisticated attacks including packet rewriting (often assumed possible but rarely demonstrated in a modern ad-hoc radio context) and man-in-the-middle.

We trust that this paper will help vendors and users strengthen the security of their real-world systems.

Acknowledgements

We thank EPSRC for funding this work as part of project EP/D076870/1[10] and the Isaac Newton Trust for co-sponsoring Matt as a UROP summer student. Dan also thanks the MSM 0021630528 project. We are grateful to our industrial partners, interviewed for the risk assessment, and to our colleagues from the Engineering Department of the University of Cambridge for support, especially Neil Hoult who also commented on the paper.

References

1. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next century challenges: Mobile networking for "smart dust". In: Proc. 5th ACM MobiCom, pp. 271–278. ACM Press, New York (1999)
2. Karlof, C., Sastry, N., Wagner, D.: Tinysec: A link layer security architecture for wireless sensor networks. In: Proc. 2nd SenSys, pp. 162–175 (2004)
3. Becher, A., Benenson, Z., Dornseif, M.: Tampering with motes: Real-world physical attacks on wireless sensor networks. In: Clark, J.A., Paige, R.F., Polack, F.A.C., Brooke, P.J. (eds.) SPC 2006. LNCS, vol. 3934, pp. 104–118. Springer, Heidelberg (2006)
4. Stajano, F., Anderson, R.: The resurrecting duckling: Security issues for ad-hoc wireless networks. In: Malcolm, J.A., Christianson, B., Crispo, B., Roe, M. (eds.) Security Protocols 1999. LNCS, vol. 1796, pp. 172–194. Springer, Heidelberg (2000)
5. Karlof, C., Wagner, D.: Secure routing in wireless sensor networks: Attacks and countermeasures. Ad Hoc Networks 1(2–3), 293–315 (2003)

[10] `http://www.winesinfrastructure.org/`

6. Mainwaring, A., Culler, D., Polastre, J., Szewczyk, R., Anderson, J.: Wireless sensor networks for habitat monitoring. In: Proc. 1st ACM WSNA, pp. 88–97. ACM Press, New York (2002)
7. Arora, A., Ramnath, R., Ertin, E.: Exscal: Elements of an extreme scale wireless sensor network. In: Proc. 11th IEEE RTCSA, pp. 102–108 (2005)
8. Werner-Allen, G., Lorincz, K., Welsh, M., Marcillo, O., Johnson, J., Ruiz, M., Lees, J.: Deploying a wireless sensor network on an active volcano. IEEE Internet Computing 10(2), 18–25 (2006)
9. Krishnamurthy, L., Adler, R., Buonadonna, P., Chhabra, J., Flanigan, M., Kushalnagar, N., Nachman, L., Yarvis, M.: Design and deployment of industrial sensor networks: Experiences from a semiconductor plant and the north sea. In: Proc. 3rd SenSys, pp. 64–75. ACM Press, New York (2005)
10. Kim, S., Pakzad, S., Culler, D., Demmel, J., Fenves, G., Glaser, S., Turon, M.: Health monitoring of civil infrastructures using wireless sensor networks. In: Proc. 6th IPSN, pp. 254–263. ACM Press, New York (2007)
11. Perrig, A., Szewczyk, R., Tygar, J.D., Wen, V., Culler, D.E.: Spins: Security protocols for sensor networks. Wireless Networks 8(5), 521–534 (2002)
12. Eschenauer, L., Gligor, V.D.: A key-management scheme for distributed sensor networks. In: Proc. 9th ACM CCS, pp. 41–47. ACM Press, New York (2002)
13. Hu, Y.C., Perrig, A., Johnson, D.B.: Ariadne: a secure on-demand routing protocol for ad hoc networks. Wireless Networks 11(1-2), 21–38 (2002)
14. Chen, B., Jamieson, K., Balakrishnan, H., Morris, R.: Span: an energy-efficient coordination algorithm for topology maintenance in ad hoc wireless networks. wireless networks 8(5), 481–494 (2002)
15. Wood, A.D., Stankovic, J.A., Zhou, G.: Deejam: Defeating energy-efficient jamming in ieee 802.15.4-based wireless networks. In: Proc. 4th IEEE SECON, pp. 60–69. IEEE Computer Society Press, Los Alamitos (2007)
16. Luk, M., Mezzour, G., Perrig, A., Gligor, V.: Minisec: a secure sensor network communication architecture. In: Proc. 6th IPSN, pp. 479–488. ACM Press, New York (2007)