Node Compromise in Sensor Networks: The Need for Secure Systems

Carl Hartung, James Balasalle, Richard Han

Department of Computer Science
University of Colorado at Boulder

Technical Report CU-CS-990-05

January 2005

Node Compromise in Sensor Networks: The Need for Secure Systems

Carl Hartung, James Balasalle, Richard Han

Department of Computer Science
University of Colorado, Boulder

{carl.hartung, james.balasalle, richard.han}@colorado.edu

ABSTRACT

While sensor network deployment is becoming more commonplace in environmental, business, and military applications, security of these networks emerges as a critical concern. Without proper security, it is impossible to completely trust the results reported from sensor networks deployed outside of controlled environments.

Much of the current research in sensor networks has focused on protocols and authentication schemes for protecting the transport of information. However, all of those schemes are useless if an attacker can obtain a node from the network and extract the appropriate information, such as security keys, from it.

We focus our research on the area of secure systems. In this paper we demonstrate the ease with which nodes can be compromised as well as show exactly what information can be obtained and how it can be used to disrupt, falsify data within, or eavesdrop on sensor networks. We then suggest mechanisms to detect intrusions into individual sensor nodes. Finally, we come up with security measures that can be implemented in future generation nodes to improve security.

1. INTRODUCTION

While node compromise is often discussed as a potential vulnerability in sensor networks [5], almost no work has been done to prove the viability of such attacks. Thus, we have designed and carried out a number of experiments detailing the relative ease with which commonly used current generation sensor nodes can be compromised using regular 'off the shelf' products and readily available, free software. While we found it is very easy to compromise a node and exploit it for various purposes, we also found a number of improvements that can be made in future generations of nodes in order to help alleviate the possibility of sensor node compromise.

To understand the dangers of node compromise, we must first define what we mean by node compromise. Node compromise occurs when an attacker, though some subvert means, gains control of a node in the network after deployment. Once in control of that node, the attacker can alter the node to listen to information in the network, input malicious data, cause DOS, black hole, or any one of a myriad of attacks on the network. The attacker may also simply extract information vital to the network's security such as routing protocols, data, and security keys. Generally compromise occurs once an attacker has found a node, and then directly connects the node to their computer via a wired connection of some sort. Once connected the attacker controls the node by extracting the data and/or putting new data or controls on that node.

Generally, all sensors must have an interface

that enables them to be programmed by another machine. The exception to this would be preprogrammed sensors for a specific task, but current sensor technology demands that sensors are flexible and can be programmed and reprogrammed for a number of different uses. Tamper proof hardware is also available, but significantly increases cost and reduces the leeway for user/programmer error, as well as eliminates the reprogramability. Typically the programming interface on a sensor node is either a direct serial, parallel connection, or USB connection (henceforth referred to as 'wired'), or an intermediate programming board which is connected to both the node and a wired connection. These programming interfaces, which provide so much flexibility in programming sensor nodes, also provide would-be attackers with the easiest means of compromise.

For our experiments we programmed a sensor node with various applications and various operating systems, and attempted to extract that information using freely available software, standard computer exploits such as buffer overflows, and various debugging mechanisms. Debugging tools are usually the most popular weapons for attackers to determine how to exploit programs, and that generality is still true for sensor networks. We tried our exploits on both TinyOS[3], an event based operating system, and Mantis OS[4], a multi-threaded operating system. Our results showed that both operating systems were susceptible to the same attacks.

In our experiments were able to, in very little time (<1 minute), extract all of the information located onboard the node's EEPROM, Flash, and SRAM. Once the attacker has access to all of this data, they can analyze it to ascertain keys, routing protocols, and other security sensitive information. With this information an attacker could adversely affect the network in a variety of ways. Attackers could simply use the keys to decrypt messages and listen to all the traffic in the network, or possibly modify the code in order to inject malicious messages into the network and confuse it, or provide false data to the end user application.

The remainder of this paper is organized as follows: Section 2 discusses the design of current sensor hardware. Section 3 analyzes different attack models and scenarios. We discuss the problems presented by these vulnerabilities in Section 4. In section 5 we discuss possible improvements for next generation nodes, and in section 6 we discuss future work which needs to be done in this area. Finally, we sum up the conclusions of our paper in section 6.

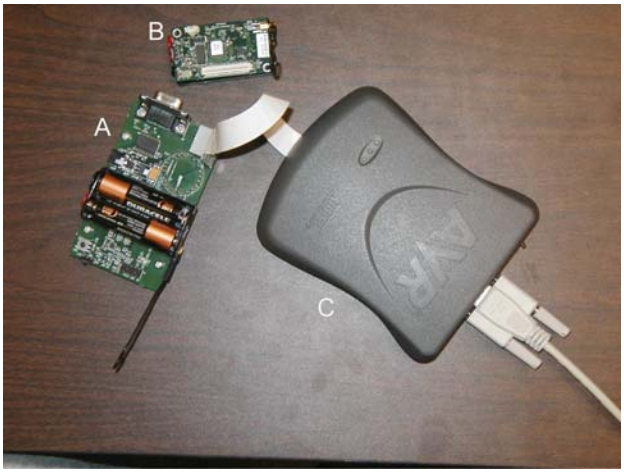

Figure 1) A) Programming board with Mica2 plugged in. B) Other side of Mica2. C) AVR ICE JTAG programmer connected to programming board with ribbon and connected to computer via serial cable.

2. SENSOR HARDWARE

The sensor hardware we chose for our experiments was the Mica2 mote by XBow. These motes are currently the most widely used sensor node, having been used in several wireless sensor network installations [2].

The Mica2 uses an Atmega128 chip for its processor. It is an 8-bit processor running at 4MHz. They come with 128KB of instruction flash, 4KB SRAM, and 4KB EEPROM. The chip is designed using a harvard architecture. It is important to note the harvard architecture as it prevents a handful of standard computer attacks, such as buffer overflow, which we will

discuss later in the paper. Interestingly enough, the use of the harvard architecture in embedded systems, and specifically in sensor nodes, was not intended to provide security features. Rather, the design of the harvard architecture, using separate memories for data and instructions and requiring different busses for each, allows instructions and operands to be fetched simultaneously. This means the architecture can run much faster as it is able to fetch the next instruction at the same time it completes the current instruction. It is important to note the improvement in speed as the nodes tend to be extremely resource constrained. Thus, though only intended to provide a speed boost to the system, the harvard architecture actually provides a good level of security against radio attacks, specifically against standard buffer overflow attacks.

The Mica2 has a serial interface connected to a programming board. Typically the Mica2 is programmed using the intermediate programming board connected to the computer with either a serial or parallel connection. The programming board also has a JTAG interface which also allows for programming, as well as using GBD for On Chip Debugging (OCD). A JTAG interface is an IEEE standardized interface to processors which allows for accessing and controlling signal levels on the chip. Figure 1 shows an AVR ICE JTAG programmer interface as well as a programming board with a mica2 attached.

Though we only tested the Mica2 hardware, our results can be generalized over sensor networks as a whole. Code on current sensor nodes needs to be installed and reinstalled. Thus, today's sensor technology requires direct access to the node via a wired interface or through some type of intermediate programming board. Also, not all processors use the JTAG interface for OCD, but most which have been used in sensor node development do, and others provide similar chip debugging mechanisms.

3. ATTACK MODELS

3.2 Types of Attack

There are different attack models we must consider, but many are outside the scope of this paper. An attacker may range from a prankster with a laptop and a serial cable with a few hours to kill to a military installation with hundreds of scientists with unlimited money and time.

We first consider the latter of the above mentioned cases: a military installation or other science lab with many scientists. In this example you can assume that the scientists have access to oscilloscopes, process analyzers, and any number of other analytical machines to help them crack the system. These machines typically cost thousands of dollars and are rarely found outside of such installations. Given the resources of such installations, it is reasonable to assume that given the correct amount of time all information could be extracted from a sensor node. The only real way to prevent this is tamper proof hardware which triggers some type of self destruct mechanism upon attempted compromise. One can also assume that if the enemy has such capabilities and desires to learn what the sensor net is doing, that the implementers would, in fact, use some sort of tamper proof hardware.

Since it is almost impossible to stop unlimited time and money, and tamper proof hardware is generally prohibitively expensive, the above scenario is a bit beyond the scope of

```
hartung@marley /opt/tinyos-1.x/apps/TestTinySec
$ make mica2 install MIB510=/dev/ttyS0
    compiling TestTinySec to a mica2 binary
ncc -o build/mica2/main.exe -Os -board=micasb -target=mica2 -I%T/lib/TinySec -I%
T/lib/Counters -Wall -Wshadow -DDEF_TOS_AM_GROUP=0x7d -Wnesc-all -DTINYSEC_KEY="
0x1B,0x6F,0xBD,0x4B,0x85,0x9A,0xBB,0x6D,0xCD,0x15,0x4B,0x93,0xF6,0xEB,0x71,0x30"
 -DTINYSEC_KEYSIZE=8 -finline-limit=100000 -fnesc-cfile=build/mica2/app.c  TestT
inySec.nc -lm
    compiled TestTinySec to build/mica2/main.exe
          16754 bytes in ROM
            840 bytes in RAM
avr-objcopy --output-target=srec build/mica2/main.exe build/mica2/main.srec
make[1]: Entering directory `/opt/tinyos-1.x/apps/TestTinySec'
    installing mica2 binary
uisp -dprog=mib510 -dserial=/dev/ttyS0 -dpart=ATmega128 --wr_fuse_e=ff  --erase
--upload  if=build/mica2/main.srec
Firmware Version: 2.1
Atmel AVR ATmega128 is found.
Uploading: flash

Fuse Extended Byte set to 0xff
make[1]: Leaving directory `/opt/tinyos-1.x/apps/TestTinySec'
```

Figure 2) Programming a node with Tiny OS' TinySec feature. Key is highlighted.

```
00000170  94 56 6d 98 9b 76 97 fc  b2 c2 b0 fe db 20 e1 eb  |.Vm..v..... ..|
00000180  d6 e4 dd 47 4a 1d 42 ed  9e 6e 49 3c cd 43 27 d2  |...GJ.B..nI<.C'.|
00000190  07 d4 de c7 67 18 89 cb  30 1f 8d c6 8f aa c8 74  |....g...0......t|
000001a0  dc c9 5d 5c 31 a4 70 88  61 2c 9f 0d 2b 87 50 82  |..]\1.p.a,..+.P.|
000001b0  54 64 26 7d 03 40 34 4b  1c 73 d1 c4 fd 3b cc fb  |Td&}.@4K.s...;..|
000001c0  7f ab e6 3e 5b a5 ad 04  23 9c 14 51 22 f0 29 79  |...>[...#..Q".)y|
000001d0  71 7e ff 8c 0e e2 0c ef  bc 72 75 6f 37 a1 ec d3  |q~.......ruo7...|
000001e0  8e 62 8b 86 10 e8 08 77  11 be 92 4f 24 c5 32 36  |.b.....w...O$.26|
000001f0  9d cf f3 a6 bb ac 5e 6c  a9 13 57 25 b5 e3 bd a8  |......^l..W%....|
00000200  3a 01 05 59 2a 46 1b 6f  bd 4b 85 9a bb 6d cd 15  |:..Y*F.o.K...m..|
00000210  4b 93 f6 eb 71 30 00 00  00 00 00 00 00 00 00 00  |K...q0..........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000280  1b 6f bd 4b 85 9a bb 6d  00 00 1b 6f bd 4b 85 9a  |.o.K...m...o.K..|
00000290  bb 6d 00 00 1b 6f bd 4b  85 9a bb 6d 00 00 1b 6f  |.m...o.K...m...o|
000002a0  bd 4b 85 9a bb 6d 00 00  1b 6f bd 4b 00 00 00 00  |.K...m...o.K....|
000002b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
```

Figure 3) Hex dump of output of SRAM gathered using the JTAG interface. Key is highlighted.

this paper. Therefore, we turn our focus to the simpler and more likely case of someone with a laptop or computer, a serial cord, and possibly a programming board. Assuming the user has a computer, serial cords are on the order of dollars, and programming boards usually run in the tens to hundreds of dollars, making this scenario much more plausible. We also assume that the attacker has good familiarity with standard debugging tools such as GDB. With these assumptions, we now show how an attacker can compromise a node in less than 1 minute.

3.2 Physical Attacks

First, we used only a programming board and a serial cable to launch our attack. Using a freely available tool called UISP we were able to dump the program flash as well as the information stored in the EEPROM. A simple execution of an avr tool, avr-objcopy, converted the source flash into an assembly file as shown in figure 4. Once in assembly format, an attacker could analyze the code to ascertain routing protocols and/or any pre-coded keys. Depending on the complexity of the program,

the analysis could take significant time, but that is outside the scope of this paper. The danger here is that it took <1 minute to obtain the source image. As all of the above commands are simply one command line execution with a handful of arguments, a majority of the time, about 45 seconds, was spent transmitting the binary image over the relatively slow serial interface onto the computer's hard drive, or converting from one format to another. The rest of the time, about 15 seconds, was human interface time typing the necessary commands for retrieving the flash, which can also be automated for maximum speed.

Next, we used an AVR JTAG interface to attempt to acquire the same data from the sensor node. With the JTAG programmer we found that not only were we able to dump the program flash and the EEPROM, but we were also able to dump the chip's SRAM in a matter of seconds. Generally the SRAM is considered the safest place to store keys and other sensitive information due to its volatile nature. However, the ease with which the data was extracted from SRAM proved that the notion of SRAM being safer is false. This alone invalidates the security claims of many of the global and shared key schemes. Again, the extraction took mere seconds and again a majority of the time was spent transferring the data from the node to the computer. The longest task was dumping the program flash (128K), and that only took ~30 seconds.

After having discovered the above simple methods to get the data, we ran an experiment to quickly analyze the ease at which we can discover keys buried within that data. We loaded TinyOS' TinySec[1] security protocol onto a node and then used our JTAG attack to determine if we could decipher the keys. We dumped the SRAM and converted it into HEX format, again using freely available tools, and saw the "secret" key right in front of us. A few iterations with other programs that have implemented TinySec showed that the key was always in the same location in SRAM. Figures 2 and 3 show the output from both TinySec's programming the key into the node, and the HEX dump of the SRAM from the node. One thing to note is that TinySec does not specify any key pre-distribution method, it merely assigns a global key to the system. Thus, an attacker seeking the TinySec key need only target a well-known address or area of memory in advance, rather than downloading the complete binary image of the operating system and applications, thereby reducing the download further and enabling compromise in mere seconds.

```
2e0:    f1 04       cpc    r15, r1
2e2:    01 05       cpc    r16, r1
2e4:    11 05       cpc    r17, r1
2e6:    68 f3       brcs   .-38        ; 0x2c2
2e8:    81 e7       ldi    r24, 0x71    ; 113
2ea:    80 93 00 01    sts    0x0100, r24
2ee:    60 e0       ldi    r22, 0x00    ; 0
2f0:    70 e0       ldi    r23, 0x00    ; 0
2f2:    82 e0       ldi    r24, 0x02    ; 2
2f4:    0e 94 7d 1c    call   0x38fa
2f8:    dc 01       movw   r26, r24
2fa:    fc 01       movw   r30, r24
2fc:    12 96       adiw   r26, 0x02    ; 2
2fe:    9c 91       ld     r25, X
```

Figure 4) Assembly output after retrieving from flash and altering using avr-objcopy.

4. DISCUSSION

Looking at most current security protocols in sensor networks reveals assumptions that keys or algorithms are hard to obtain. Our research shows the opposite of this assumption. Given that it takes less than 1 minute to dump all of the EEPROM, program Flash, and a chip's SRAM, it is impossible to assume that any stored keys are safe on a sensor node. Also, given enough time assembly code can be analyzed and modified, or even decompiled into C code using various primitive decompilers.

Since we cannot truly state that our keys on our nodes are secure, we cannot then state that

our data is secure an accurate. This could lead to huge issues in any sensor node deployment from agriculture to military. Even our supposedly secure protocols cannot be considered safe since one would never know if an attacker had obtained the protocol's keys or not.

These discoveries demonstrate the need to develop secure systems in sensor networks. Simply securing the data transmitted from node to node is not enough. If a single node in the network can be compromised, the security encapsulating the transmitted data is also compromised. Though tamper proof hardware is available, it is too expensive to use in most deployments. Therefore, securing the entire system is essential to guarantee accurate and secure data.

As is, sensor nodes cannot determine whether a user is simply using debugging tools or attempting to hack in. There need to be mechanisms to turn off debugging tools at a level other than hardware. These problems are not paramount in standard computer systems as it is rare that someone will open a computer up and directly interface with its processor or motherboard. However, since direct communication is such a common way of interfacing with sensor nodes, the ability to disable debugging tools is important to ensure security.

Current literature[6,13] suggests schemes where pre-distributed keys are erased after new keys have been established using the pre-distributed ones. We agree that these approaches mitigate node compromise. These schemes usually involve encrypting a challenge with the key before destroying it. This challenge enables the node to communicate with other nodes that also have the challenge. Since the challenge is stored in memory, compromising the node still gains access to the challenge. The literature[9] suggests ways to find keys in memory, but since a challenge could be vastly different from a key, it might take significantly longer to find. Therefore erasing a node's keys could significantly slow down an attacker, but since that attacker can access all the SRAM it may not completely prevent an attacker from still participating in the network.

LEAP [13] assumes that a global key is erased after an initial setup period T_{est}. The assumption is that T_{est} is much less than the time to compromise a node T_{min}. However, as we have shown, this assumption can be violated if it only takes on the order of seconds to compromise a node. Also, in this scheme, we would expect that there will be cases when T_{est} would in reality be on the order of tens of minutes in certain deployment schemes, e.g. dropped and scattered from airplanes. In these scenarios, the scattered nodes, even if dropped simultaneously, may arrive in different parts of the network at different times and will need some slack time to set up the network and bootstrap pairwise links using the transitory global key. During this time, if an adversary observes a node and quickly obtains the key using any of the techniques shown here, i.e. $T_{est} > T_{min}$, then the global key will be compromised, allowing the adversary unlimited access to any portion of the network. LEAP also assumes that moving the global key from non-volatile memory into volatile memory provides added security. As we have shown, that assumption is false, because both RAM and flash are accessible to an adversary.

5. IMPROVEMENTS

As stated above the Atmel processor has an on chip debugging feature. This feature is what enables us to easily obtain the contents of main memory. On chip debugging greatly facilitates the development of new applications and devices. However, it exposes a new set of security vulnerabilities to an attacker. Currently it is possible to turn the on-chip debugging feature off. However, it is a very simple to turn it back on using publicly available debugging tools. A version of the Atmel processor with the on-chip debugging feature on/off switch in software rather than hardware would eliminate a category of

possible attacks. Furthermore, if the OCD request generated an interrupt which could be caught by software then the node could erase any important information. Toggling the OCD in software would mean that an attacker would have to replace the code image on the node, destroying all of the data they are trying to extract. Sensor networks deployed in especially hostile environments such as a battlefield, or in particularly sensitive areas such as a hospital or financial applications, it would be desirable to have a sensor node which would not respond to the standard on-chip debugging.

Another possible solution would be to use location aware applications[8] that could detect movement on a fine scale, GPS, or group communication techniques. The network could then mark 'moved' nodes as possibly compromised and flag their data at the end-user application. Furthermore, if a node can detect its own movement by either accelerometers or GPS then it can preemptively delete important information stored in SRAM, flash, or anywhere else on the system.

6. FUTURE WORK

Future research in this area needs to be done so that we can understand better ways of preventing and detecting system level attacks. Possible areas of prevention include location aware nodes that can detect when they are moved. Another means of prevention is the hardware support to disable the on-chip debugging, which would prevent an attacker from using a JTAG or similar device. While research on intrusion detection of a network is underway, intrusion detection of an individual node is an area of research that especially needs to be addressed. Intrusion detection is extremely difficult because of the resource constraints imposed by sensor node hardware. In SWATT [12], nodes apply a MAC to the operating system to detect whether the binary image of the operating system has been changed and new code loaded. Since an attacker can completely erase and reprogram a node, it is difficult to detect this behavior, especially when the "new" node still contains all the required security information. When an attacker physically finds a node without tamper resistant hardware, he has fewer constraints and many attack options are available to him. However, this means he has to find the nodes first. Finally, one more area that needs additional research is that of public key infrastructure on sensor nodes. In [7] elliptic curve cryptography is presented to address some of the shortcomings of most key pre-distribution solutions. Developing this idea could also better secure the operating systems of sensor nodes.

7. CONCLUSIONS

There is a great need to design secure systems for sensor networks. The flexibility of the current generation of sensor nodes leaves too many holes open which allow attackers access to vital system information. Until such systems exist, it is impossible to confidently trust the data from any sensor network deployed outside of a controlled environment. We have shown that it is trivially easy to retrieve program code, static data, and even dynamic program memory from sensor nodes. Current sensor nodes are easily tampered with, code can be easily altered, and system critical information is easily obtained using freely available software and cheaply available hardware. With this information, attackers have virtually free reign to spy on, participate in, or subvert sensor networks. There exist several research opportunities to pursue in the directions of detecting attempted compromise, or outright prevention of node compromise. Only when we have a secure system design can we be confident that our secure transmission protocols will once again be safe.

8. BIBLIOGRAPHY

[1] C. Karlof, N. Sastry, D. Wagner, "TinySec: A Link Layer Security Architecture for Wireless Sensor Networks", to appear in ACM SenSys 2004

[2] A. Mainwaring, J. Polastre, R. Szewczyk D. Culler, J. Anderson,"Wireless Sensor Networks for

Habitat Monitoring", First ACM Workshop on Wireless Sensor Networks and Applications (WSNA) 2002, pp. 88-97.

[3] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister . "System Architecture Directions For Network Sensors", ASPLOS 2000.

[4] H. Abrach, S. Bhatti, J. Carlson, H. Dai, J. Rose, A. Sheth, B. Shucker, R. Han, "MANTIS: System Support for MultimodAl NeTworks of In-situ Sensors", 2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA) 2003, pp. 50-59.

[5] L. Eschenauer and V. Gligor. "A key management scheme for distributed sensor networks." In Proceedings of the 9th ACM Conference on Computer and Communication Security, pages 41–47, November 2002.

[6]D. Liu, P. Ning, "Establishing Pairwise Keys in Distributed Sensor Networks," in Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS '03), pages 52--61, Washington D.C., October, 2003.

[7] D. Malan, M. Welsh, M. Smith, "A Public-Key Infrastructure for Key Distribution in TinyOS Based on Elliptic Curve Cryptography", IEEE SECON 2004.

[8] Andreas Savvides, Chih-Chieh Han and Mani B. Strivastava. "Dynamic fine-grained localization in ad-hoc networks of sensors." 7-th annual international conference on Mobile computing and networking (MobiCom) 2001, July 16 - 21, 2001, Rome Italy. Pages 166-179.

[9]Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, "Data Lifetim is a Systems Problem", To appear in the 2004 SIGOPS European Workshop.

[10] Jonathan Hui, David Culler, "The Dynamic Behavior of a Data Dissemination Protocol for Network Programming at Scale", Sensys '04

[11] B. Przydatek, D. Song, A. Perrig, "SIA: Secure Information Aggregation in Sensor Networks", ACM SenSys 2003

[12] A. Seshadri, A. Perrig, L. van Doorn, and P. Khosla. Swatt: Software-based attestation for embedded devices. In Proceedings of the IEEE Symposium on Security and Privacy, May 2004.

[13] S. Zhu, S. Setia, and S. Jajodia. Leap: Efficient security mechanisms for large-scale distributed sensor networks. In 10th ACM Conference on Computer and Communications Security, Washington D.C, USA, October 2003.