Extracting Keys from Second Generation Zigbee Chips

Travis Goodspeed
1933 Black Oak Street
Jefferson City, TN, USA
travis@radiantmachines.com

ABSTRACT

First generation Zigbee chips were SPI slaves with no internal processing beyond cryptographic acceleration. Extracting a key was as simple as spying on the SPI transactions. The second generation chips, typified by the CC2430 from Texas Instruments and the EM250 from Ember, contain both a microcontroller and a radio, making the SPI sniffing attack all but irrelevant. Nevertheless, both chips are vulnerable to local key extraction. This paper describes techniques for doing so, focusing on the CC2430 as the EM250 has no protection against outside access. Recommendations are made for defending CC2430 firmware by using compiler directives to place sensitive information in flash memory, rather than in RAM. All Chipcon radios with 8051 cores released prior to the publication of this paper are expected to be vulnerable.

Keywords

Zigbee, CC2430, EM250, System on a Chip (SoC)

1. GENERATIONS

First generation Zigbee chips, such as the CC2420, were simply digital radios with SPI interfaces and a bit of hardware-accelerated cryptography. They could not run a Zigbee stack themselves, but rather relied upon an external microcontroller to perform even the most basic of functions. It should also be made clear that Zigbee is one specification for the use of these radios, which in the first generation never implemented more than the lowest layers of IEEE 802.15.4. [7] pointed out the obvious with regard to the first generation, which is that keys sent as cleartext by SPI can be sniffed to allow an attacker to participate in a network.

The second generation of Zigbee chips were the first to hold a complete Zigbee implementation internally, thanks to the addition of internal, reprogrammable microprocessors. They also lack the vulnerability to bus probing, as keys need not travel over an exposed SPI bus. That said, the microcontroller cores were added for convenience, not security, as will be explained below.

The third generation of chips will include more powerful microprocessors and–hopefully–a lot more security. The offering from Texas Instruments is the CC430 family, based upon the MSP430X2 processor. Ember will be using the Arm Cortex M3 in its EM300 series. These chips are out of scope for this paper, as they are not yet commercially available. Also, Freescale's line of radios have not yet been examined by the author, but they will be in the near future.

2. CONCERNING THE EM250

The Ember EM250 contains a 16-bit XAP2b microprocessor from Cambridge Consultants Ltd.[3] Debugging support is provided by that firm's proprietary SIF protocol, with hardware and software available only through Ember. SIF itself is a variant of JTAG.

While the datasheet and various piece of marketing literature claim "The EM250 employs a configurable memory protection scheme usually found on larger microcontrollers.", this refers not to a debugging fuse or bootloader password, but rather to protection from accidental self-corruption of memory. This is in the form of Application/System separation, allowing the EmberZNet stack to defend certain regions of RAM and radio registers from accidental corruption by the application software.[3]

The author has not yet evaluated the protection scheme for its use in defending against stack overflow exploits. It is possible, but not likely, that local privilege escalation from Application to System mode is difficult.

In any case, the SIF debugging port of the EM250 does not contain a security fuse. There is no supported method of denying access to an attacker who controls those pins. Ember is aware of the oversight, and they expect that the third generation EM300 series does not share this vulnerability. There are presently no plans to fix the EM200 series.

3. CONCERNING THE CC2430

The TI/Chipcon 2430 is a the combination of Chipcon's radio technology with an 8051 microcontroller. Also in the same family are the CC1110, CC2431, CC2510, CC2511, CC2530, and CC2531. These other chips have not been tested by the author, but are expected by the documentation to be similarly vulnerable at the time of this writing.

To appear at Black Hat USA 2009
July, 2009, Las Vegas, Nevada.

Like many schemes, including the bootloader (BSL) of some MSP430 microcontrollers [6][5], the CC2430 protects flash memory but not RAM. Further, as a Harvard-architecture chip, all constants are copied into RAM by default as a performance feature of the compiler. Thus, while the protection scheme of the chip is sufficient to protect its programming, it does nothing to prevent the extraction of keys! Extraction is as simple as erasing the chip by the debugger, then reconnecting and dumping RAM.

4. EXPERIMENTAL APPARATUS

The GoodFET[1] is a USB bus adapter developed by the author as a means of learning, among other things, the JTAG protocol. Its firmware includes support for the Chipcon debugging protocol, as documented in [1], allowing the Chipcon radios to be debugged using a python script, "goodfet.cc".

This vulnerability can be tested for on any 8051-based Chipcon device using the script provided in Figure 2. The script first places 32 bytes of data into RAM, then dumps those bytes back out as *foo.hex*. It then executes CHIP_ERASE to clear any security fuses and erase flash memory before dumping data a second time as *bar.hex*. If the two dumps are identical, then Data memory (RAM) has not been erased along with Code memory (Flash). If the two differ, then the chip is not vulnerable to this particular attack.

Figure 1 demonstrates that the CC2430 revision 0x04 is vulnerable. Similar results are obtained for all other chips of this family at the time of this writing. Chipcon has been made aware of the vulnerability, and they intend to fix future chips.

5. PROTECTING 8051 CONSTANTS

Key identification within non-key data has been demonstrated by various authors. The authors of [4] manage to reliably identify disk encryption keys despite from briefly unrefreshed DRAM, and they conclude with the statement that

> Ultimately, it might become necessary to treat DRAM as untrusted, and to avoid storing sensitive information there, but this will not become feasible until architectures are changed to give software a safe place to keep its keys.

While personal computers and the EM250 might lack such a place, it is possible to instruct an 8051 compiler to store a constant in Code (Flash) memory, rather than in Data (RAM). This is described in [2], but as a workaround for RAM limitations rather than as a security measure.

The "__code" keyword must be applied to any *const* variable within code memory, as well as any pointer to such a constant. This is because the 8051, as a Harvard machine, does not have a unified address space. Also note that there is a slight performance penalty to fetches from code memory, as they cannot occur at the same time as an instruction fetch.

[1] http://goodfet.sf.net/

6. CONCLUSION

This paper has shown that all Chipcon radios at the time of publication are vulnerable to key theft because of unprotected Data memory. Further, as all popular 8051 compilers place even constants in Data memory for performance reasons, it can be assumed that all products which were shipped prior to the authorship of this paper are vulnerable. Extracting a key is as simple as connecting a debugger, erasing the chip, then freely reading the contents of RAM.

Further, as the competing radios from Ember offer even less security, the tamper resistance of wireless sensors should perhaps be considered forfeit by default.

7. REFERENCES

[1] Chipcon. CC1110/CC2430/CC2510 debug and programming interface specification.

[2] Chipcon. DN200 using constants in Code with Z-stack.

[3] Ember. EM250 datasheet.

[4] J. A. H. et al. Lest we remember: Cold boot attacks on encryption keys.

[5] T. Goodspeed. MSP430 BSL passwords: Brute force estimates and defenses, June 2008.

[6] T. Goodspeed. A side-channel timing attack of the MSP430 BSL. Black Hat USA, August 2008.

[7] T. Goodspeed, D. Highfill, and B. Singletary. Low-level design vulnerabilities in wireless control systems hardware. In *S4 2009*.

APPENDIX

A. ACKNOWLEDGMENTS

It should be noted that Ember was extremely quick to reply when contacted about the issue presented within this paper. While TI was not so quick to respond, thanks are due to those employees who ensured that the issue was acknowledged.

```
Chipcon erasure vulnerability test.
by Travis Goodspeed

This script uses the GoodFET to load 32 bytes into data memory
then it sends CHIP_ERASE before requesting those same bytes back.
If the two files are identical, CHIP_ERASE has not cleared data
memory and keys held in data memory are exposed to an attacker.

1) Writing something non-random into data memory.
2) Dumping it back out, so we can compare by MD5 checksum.
3) Erasing the contents of memory, including fuses, by CHIP_ERASE.
4) Dumping a second time, now that the chip has been erased.
5) Comparing:
180e42d3d3850b7a31636385dc2c2eee  foo.hex
180e42d3d3850b7a31636385dc2c2eee  bar.hex
Images match.  RAM is not cleared during a CHIP_ERASE.
```

Figure 1: Demonstration Log

```
echo "1) Writing something non-random into data memory."
goodfet.cc writedata app.hex 0xffe0 0xffff >>/dev/null
echo "2) Dumping it back out, so we can compare by MD5 checksum."
goodfet.cc dumpdata foo.hex 0xffe0 0xffff >>/dev/null
echo "3) Erasing the contents of memory, including fuses, by CHIP_ERASE."
goodfet.cc erase >>/dev/null
echo "4) Dumping a second time, now that the chip has been erased."
goodfet.cc dumpdata bar.hex 0xffe0 0xffff >>/dev/null
echo "5) Comparing:"
#Print results.
md5sum foo.hex bar.hex && \
echo "Images match.  RAM is not cleared during a CHIP_ERASE." || \
echo "Images differ.  Vulnerability not confirmed."
```

Figure 2: Vulnerability Test Script