Firmware Protection and Attacks Against the ATMega Microcontroller Series

Dionisio Perez-Mavrogenis

March 15, 2014

Abstract

The most abstract abstract of some abstracts, full of abstractions.

1 Introduction

This paper will present an overview of the current attacks and methods of tampering with Intellectual Property (IP) in MicroController Units (MCUs). In this context IP does not solely refer to the firmware running on the MCU but it includes information that could be obtained from that code, i.e. secrets or proprietary algorithms that help a manufacturer achieve higher performance over their competitors.

Tampering (or theft) detection and prevention of the firmware of a MCU is a popular problem with active research, as solving it would benefit a lot of parties (including the government and the military), as MCUs are used in areas spanning from consumer electronics to missile guiding systems.

A distinction between ordinary and secure microcontrollers should be made[5] and, due to the sophistication in the protective mechanisms and the attack vectors, a broad classification of attackers as[1]:

- **Home Hackers** People who are curious about the inner workings of a device and a very limited budget, perhaps with limited knowledge and no malicious intent, but with no time-limit.
- **Semi-Professional Crackers** Professionals skilled on electronics, usually with access to specialised equipment and resources. Their funding might be limited, malicious intent is unclear and they might be constrained in the time they have available.
- **Funded Organisations** People or organisations with access to the same equipment as the MCU manufacturer. Their funding is usually unconstrained, there usually exists malicious intent and their time schedule is usually tight.

Firmware tampering or theft has a number of consequences. The most obvious consequence is an attacker downloading the code from a MCU and flashing it onto a MCU that they sell, effectively avoiding development and testing costs but still offering the same product as other manufacturers([2]). A less obvious, but perhaps more important, case is the case of back-dooring[1] a MCU by re-flashing on it a modified version of the firmware with coded added by the attacker in order to accomplish their malicious intents, which could have disastrous consequences if these MCUs were used for military or other sensitive operations.

1.1 Objectives

The aims of this paper are to review the possible attacks against the ATmega series of MCUs

[1]The act of adding code to a system without the user's knowledge or approval, usually to accomplish nefarious tasks.

and provide possible countermeasures or possible methods of hardening a system.

Section 2 will provide an overview of the ATmega series of AVRs and explain the most important hardware architecture aspects and protection mechanisms they offer.

Section 3 will give a (brief) overview of the current attack techniques used to override currently implemented protection mechanisms, an overview of whom is given in Section 4.

In section 5 the current attacks will be related to the ATmega, by presenting the attack vectors in more detail as well as providing references to relevant work. Section 6 will conclude the paper with a discussion on the usefulness of hardening the ATmega, contrasting that with using a MCU that is designed to be secure.

2 The ATMega MCU Series

The ATmega series of MCUs is a popular microcontroller offering a combination of a reasonably powerful MCU at a low price. They offer a variety of clock speeds and memory capacities and large operating voltage windows, hence making them both power efficient as well as enabling them to interact with peripherals with different operating voltages[2][3].

Furthermore, developing software for an AVR board is very easy, a further incentive to use an AVR over other popular options. In particular, the AVR MCUs benefit from the free `avr-libc` high-performance C runtime library(optimised for the AVR RISC architecture), the `avr-gcc` and `avr-gdb` compiler and debugger(both based on very popular and high quality GNU software tools), the `avrdude` programming software(or Atmel's proprietary `AVRStudio`) and `Simulavr` simulator software. In addition to these community tools, Atmel provides proprietary APIs for interacting with the AVR and the developers can choose from a wide variety of programmer units available for working with the AVR MCUs[6].

[2]info:https://www.newbiehack.com

[3]info:http://www.atmel.com/v2PFResults.aspx

2.1 Atmel AVR Series

The Atmel AVR series is a RISC based MCU family that consists of the ATtiny, ATmega and ATxmega sub-families as well as derivatives of the above, including 32-bit AVRs and application specific FPGAs[4].

2.2 ATMega Architecture and Features

2.2.1 Background and Features

The ATmega series of MCUs is a relatively large family of microntroller units, with the focus of this paper being on the ATmega644 and ATmega1284. The only differences between then 1284 and the 644 is that the 1284 has got more memory available and an extra timer. A summary of (some) of the features of the two units is give in Table 1[3].

Both MCUs are an enhanced RISC Harvard architecture 8-bit CPU. Figure 2 shows the conceptual difference between a Von Neuman (most modern PCs) and a Harvard architecture, where the key distinction lies in the separation of application code and program data into different memory sections (Harvard) and tasking the CPU with distinguishing between code and data that lives in the same memory region (Von Neuman). The 644/1284 implement a Harvard architecture for both power and computational efficiency, as they are able to access more than one registers simultaneously (due to the physical wiring of the CPU) and hence are able to execute an instruction per cycle as once an instruction is executing the next one is prefetched and decoded. Their operating voltages can vary between 1.8V and 5.5V (maximum operating frequency 20 MHz).

2.2.2 Memory Organisation

The 644/1284 are equipped with an EEPROM, flash memory, SRAM, a large number of general purpose registers and a large number of I/O registers (in order to be able to perform I/O) and all memory (including I/O memory

[4]info:AVR family link

mapped images) is linear, i.e. it follows the flat memory model.

The flash memory is separated into two regions, the bootloader section and application code section. The boundary between the two sections can be configured by programming the appropriate fuses, and the page size can also be configured that way as well. Both sections hold code, however code residing in the bootloader section can execute a special instruction (`SPM`[5]) which allows the bootloader code to write to *any* section in the flash memory and hence possibly modify itself(designed for purposes such as firmware upgrades). The bootloader code can be triggered by a direct jump from the application section or by programming the reset vector via the reset fuse to point to the appropriate section of the bootloader code.

The EEPROM is memory for data that needs to persist between reboots of the MCU and hence it is (widely) used to hold configuration variables and other non-temporary preferences the application code (or the bootloader) may need, having an average lifespan is 100,000 write cycles per page.

The SRAM is volatile storage and is used as the stack and heap for the software (either application code or bootloader code) as well as for storing the Register File (i.e. the 32 GP registers) I/O and Extended I/O Memory. The reserved register locations exist in order to support the use of peripheral units as well as hold program status information (e.g. the Stack Pointer can be found in one of the GP registers).Figure 1 gives an overview of the SRAM hierarchy, which is slightly different (in terms of region sizes) for the 644 and the 1284 as the 1284 offers more SRAM.

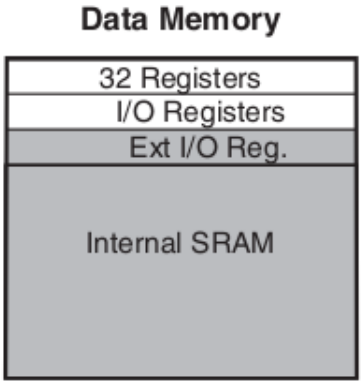


Figure 1: SRAM layout for the ATmega 644 and 1284. **Source**:[3].

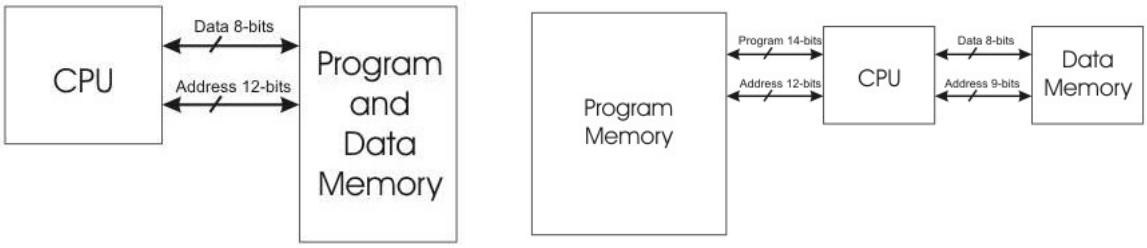
2.3 ATMega Security Features

The AVR ATmega644/1284, even though they are not meant to be trusted or secure hardware, posses certain security features. In particular, each board provides six Lock bits (which can be programmed or unprogrammed) and which are responsible for controlling or preventing different memory portions of the board to be modified or read by the other parts (e.g. prevent code executing from the bootloader section to read/write the application code section via the `SPM` instruction). The prevention however is not permanent, as that would limit the usefulness of the MCU and therefore one has the option to bring these lock bits back to State 1 (i.e. unprogrammed, having no protection scheme enabled) by issuing a Chip Erase command, which has the effect of completely erasing the Flash, EEPROM and Lock bits.

The erasing is performed with the sequence of events presented above and this is important, as one does not want to remove the access protection before removing all sensitive data and hence the Lock bits are set to 1 only after the whole program memory has been erased. Even though the flash memory has an average lifespan of 10,000 write cycles (as well as programming being relatively expensive as an operation) this approach makes sense as the ultimate goal is to preserve the intellectual property on the board rather than the board itself.

Table 2 provides an outline of the available Lock bits provided by the ATmega series. The functionality of the BLB1 group is to control access and modification of the bootloader section, group BLB0 bits control access to the application code section and group LB bits are responsible for controlling modifications on the EEPROM and Flash.A detailed explanation of their functionality and how to use them is given

[5]SPM = Store Program Code



(a) Schematic of a Von Neuman architecture.

(b) Schematic of a Harvard architecture.

Figure 2: A comparison of different machine architectures. **Source**:[4].

Model	**EEPROM (Kb)**	**SRAM (Kb)**	**Programmable Flash(Kb)**	**GP Registers**	**SPI Port**	**JTAG Interface**
ATmega644	2	4	64	32	Yes	Yes
ATmega1284	4	16	128	32	Yes	Yes

Table 1: Specification overview for the AVR ATmega644 and ATmega1284

Lock Bit Byte	**Bit Number**	**Description**	**Default Value**
BLB12	5	Boot Lock Bit	1
BLB11	4	Boot Lock Bit	1
BLB02	3	Boot Lock Bit	1
BLB01	2	Boot Lock Bit	1
LB2	2	Lock Bit	1
LB1	1	Lock Bit	1

Table 2: Security lock bits offered by the ATmega644 and ATmega1284.

in [3].

3 Attacks on Hardware

3.1 Introduction

apparently sergei has cracked them already [5] mention attacks may be passive(observing input->output mapping) or active (tamper with the aforementioned mapping in some useful way)

3.2 Non-Invasive Attacks

3.3 Semi-Invasive Attacks

all memory types are linear(as well as memory mapped IO) - related to memory scanning attacks by glitching and power faults (stop making call or jump instructions)

3.4 Invasive Attacks

overview of attack categories [each one to the category that it corresponds above]

- microprobing
- side-channel attacks
- software attacks (exploit communication protocols or crypto implementation and such)
- reverse engineering of hardware
- fault generation (power/clock glitches)

* for each category discuss budget/tools/skillset/time required

4 Countermeasures to known attacks

- overview of most popular techniques
- benefits and how they improve the situation/approach the problem
- added cost for this investment (in terms of hardware and money, transparency to the developers, runtime overhead etc)

perhaps review some popular secure chips ??

5 Securing the mega

5.1 Working attacks against the ATmega

5.2 Motivation

5.3 Current Attack Vectors

5.4 Protective Steps

* feasible? * added cost (in terms of $$, extra hardware and software implementation penalties/overhead)

6 Evaluation

No system is unbreakable and the only hope is to make it as hard to break as possible, said by [2],[5] not sure if the subsections are needed here

6.1 Attacks and Solutions overview

6.2 Conclusions

References

[1] Ross Anderson and Markus Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the 2Nd Conference on Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, WOEC'96, Berkeley, CA, USA, 1996.

[2] Atmel Corporation. Atmel AVR231: AES Bootloader. Application Note 2589E-AVR-03/12, Atmel Corporation, 2012. `www.atmel.com/Images/doc2589.pd`.

[3] Atmel Corporation. ATmega164PA/324PA/644PA/1284P Datasheet. Datasheet 8272E-AVR-04/2013, Atmel Corporation, 2013. https://secure.ecs.soton.ac.uk/notes/comp2215/rscs/ATmega1284P-datasheet_8272E-04-2013.pdf.

[4] Philipp Hof. A Primer To Microcontrollers. http://www.elec.canterbury.ac.nz/PublicArea/Staff/hof/p10-embed/p10-tutorial/index.html, 2010.

[5] Sergei Skorobogatov. Breaking Copy Protection in Microcontrollers. http://www.cl.cam.ac.uk/~sps32/mcu_lock.html, 2000.

[6] Alan Trevennor. *Practical AVR Microcontrollers: Games, Gadgets, and Home Automation with the Microcontroller Used in the Arduino (Technology in Action).* Apress, 2012.