Firmware Protection and Attacks Against the ATmega Microcontrollers

Dionisio Perez-Mavrogenis

March 21, 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 MCUs should be made[8] and, due to the sophistication in the protective mechanisms and the attack vectors, a broad classification of attackers as[1]:

- **Home Hackers** Clever and curious people with a very limited budget, perhaps with some degree of knowledge and no malicious intent, but with no time-limit.

- **Semi-Professional Crackers** Professionals skilled on electronics with access to specialised equipment and resources. Their funding might be limited, malicious intent is unclear and they might be constrained in their time allowance.

- **Funded Organisations** Organisations with access to MCU manufacturing equipment. 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 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

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

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 AVR MCU Series

The Atmel AVR series is an enhanced-RISC MCU family that consists of the ATtiny, ATmega and ATxmega sub-categories and derivatives of the above, including 32-bit AVRs and application specific FPGAs[2]. The models have varying degrees of hardware capabilities and large operating voltage windows in order to accommodate demand and integrate well with peripherals[3][4].

Developing software for an AVR is easy as the AVRs benefit from the free `avr-libc` high-performance C run-time 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. Additionally, 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 AVRs[9].

2.1 ATMega Architecture and Features

2.1.1 Important Feature Overview

The ATmega series of MCUs is a relatively large family of MCUs and the focus of this paper is 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 hardware extra timer. A summary of (some) of the features of the two units is give in Table 1[3].

[2]info:AVR family link

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

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

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, being designed to access more than one registers simultaneously (due to the physical wiring of the CPU), enabling them to execute an instruction per cycle. Their operating voltages can vary between 1.8V and 5.5V (maximum operating frequency 20 MHz).

2.1.2 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 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

[5]`SPM` = Store Program Code

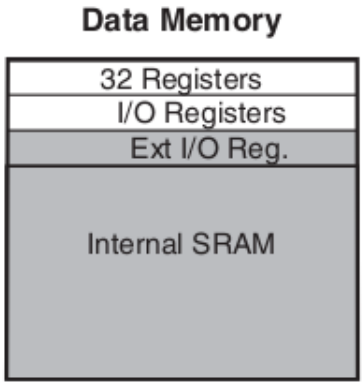

Figure 1: SRAM layout for the ATmega 644 and 1284. **Source**:[3].

Model	EEPROM (Kb)	SRAM (Kb)	Flash (Kb)
ATmega644	2	4	64
ATmega1284	4	16	128

Table 1: Specification overview for the AVR ATmega644 and ATmega1284

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.

2.2 ATMega Security Features

The AVR ATmega644/1284, even though not meant to be secure hardware modules, posses certain security features. In particular, each board provides six Lock bits responsible for controlling access to the board's memory and prevent reading or modifying the memory (e.g. prevent code executing from the bootloader section to read/write the application code section via the `SPM` instruction). This access control is not permanent, as that would limit the usefulness of the MCU and therefore one has the option to reset the lock bits (i.e. having no protection scheme enabled) by issuing a Chip Erase command, which has the effect of completely erasing the Flash, EEPROM and Lock bits.

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

Table 2: Security lock bits offered by the ATmega644 and ATmega1284. BLB stands for Boot Lock Bit and LB for Lock Bit.

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 in [3].

3 Attacks on Hardware

A distinction between *passive* and *active* attacks should be made. In the former the attacker simply monitors the chip's normal operation and tries to infer the input-output mapping whereas in the latter case the attacker actively manipulates either the chip or its operating environment with the aim of obtaining insight on the chips inner workings.

Attacks on MCUs may attempt to recover a number of artefacts, including cryptographic keys the firmware and do not need to necessarily attack the hardware itself but can exploit flaws in algorithmic

(a) Schematic of a Von Neuman architecture.

(b) Schematic of a Harvard architecture.

Figure 2: A comparison of different machine architectures. **Source**:[5].

design and implementation and protocol failures or inter-component communication patterns[1][7], obtain information by corrupting the memory or exploiting memory remanence[8][4].

3.1 Non-Invasive Attacks

Non-invasive attacks are attacks which require no de-packaging or special preparation of the chip and hence attacks under this category leave little tamper evidence behind. These attacks might be very time consuming and are not guaranteed to be successful, but are very easy and cheap to replicate once found. Furthermore, non-invasive attacks could target badly implemented communication or security protocols in order to bypass security restrictions.

3.1.1 Power Analysis

Different instructions executing on a CPU require different amounts of power, and hence one can infer which instruction is executing on a CPU by analysing a power trace generated by the MCU. These attacks are easy and relatively inexpensive to perform as they only require widely available tools.

Simple Power Analysis(SPA) involves direct observation of the MCU when it performs cryptographic operations and can leak information about both the keys and the cryptographic operations themselves (i.e. nature or structure of the algorithm).

Differential Power Analysis(DPA) extracts sensitive information by using statistical techniques on very large traces. The techniques involves obtaining power traces of known cipher-texts(but not necessarily knowing the cipher-texts) and individual bits of the key are recovered by analysing the differences in power consumption[7].

One can generally avoid noise in their power measurements by sampling the voltage (usually) on the ground line.

3.1.2 Glitch Attacks

Power glitches and *clock glitches* aim to make the CPU skip or execute incorrect instructions by applying transients. This attack can target in individual components of an MCU and a systematic search can deduce which components are affected by a given glitch sequence.

Clock glitches involve increasing the clock signal frequency so that some flip flops sample their input before being updated and hence report an incorrect value. Clock glitches are mainly aimed against software-based protection mechanisms, affecting CPU operation by supplying the CPU with incorrect data.

Power glitches work by supplying either too much power or too little, shifting transistors' threshold and causing flip-flops to read their state incorrectly. Power glitches need to be carefully synchronised with the internal clock and prolonged attacks might damage the board.

3.1.3 Data Remanence

Prolonged exposure of SRAM cells to the same values can make the cells 'remember' their state, due to material properties and stress[4]. If for example a startup routine always writes security keys to the same memory region, after some time they key will be recoverable by looking at the physical state of

the memory. Furthermore, data can be recovered from SRAM by cooling it down for a period of time after power is removed.

EEPROM suffers as well, but to a lesser extent, as material-wise one can only tell virgin-cells from used cells[8].

3.1.4 Timing Attacks

Timing attacks exploit the software implementation of cryptographic algorithms. Compiler optimisations (avoiding unnecessary branches, register and cache usage) and other implementation choices make the execution time of an algorithm dependent on the input and the secret key, rather being fixed for any input. For example, when input is compared byte-wise with a key and rejected when the first non-matching byte is found, rather than first consuming the whole input string.

Different instructions take different time to execute(e.g. `MOV eax,[eax]` is considerably slower than `INC eax`) and thus one could collect timing information for various input messages and systematically deduce the correct key.

If timing information is correlated with power analysis then defences such as constant instruction execution time could be defeated. One might use `NOP`s in the case of a wrong key in order for rejection and confirmation responses to have constant execution time but `NOP` consumes substantially less power than `INC eax` and correlating timing and power consumption information would reveal this.

3.2 Semi-Invasive Attacks

some sort of fault injection? 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)

3.3 Invasive Attacks

mention decaping and chip exposure. can be done by sending to hardware failure test labs[6] or by chemicals etc [1] Microprobing overview of attack categories [each one to the category that it corresponds above]

- microprobing
- reverse engineering of hardware

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

4 Countermeasures to known attacks

4.1 Physical Protection

protective metal layer, shrink things, mesh of wires, encase in epoxy, make layers destroy each other if removed

4.1.1 Protection circuits

radiation/temperature/voltage/frequency detector circuits that cause reset on abnormality detection (instability reference [1]). add transistors on top to hide true signal (provide reference) keep keys in own, self powered module.

4.2 Side-channel Protection

decrease signal/noise ratio (either by introducing noise or making the signal smaller), constant time/power operatations, insert random delays[8], [7] and shielding the device.

Planarization defeats optical inspection with microscope (source: Introduction to hardware security and trust, sergei skorobogatov paper).

- 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 ?? IBM 4758 is a secure device [6], some Dallas chips and perhaps more.

[6] http://www.cl.cam.ac.uk/ rnc1/descrack/ibm4758.html

5 Attacking The ATmega

The ATmega is a typical MCU that is susceptible to a successful attack by the **Home-Hacker** class of attackers due to its weak protective mechanisms. While there is a way to deliver firmware updates to the MCU securely[2], keeping the firmware secure on the device is impossible.

The motivation behind attacking the ATmega 688/1284 is their popularity and to prove how easy it is

5.1 Motivation

5.2 Attack Overview

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

6 Evaluation

No system is unbreakable and one can only harden their system enough to make the effort of breaking it unbearable to those they wish to protect against[1][8].

security is hard to do because one must carefully analyse and model all threas [7] not sure if the subsections are needed here

6.1 Securing the mega

paragraph talking about why it's not a good idea to salvage the mega.

6.2 Attacks and Solutions overview

6.3 Conclusions

References

[1] Ross Anderson and Markus Kuhn. Tamper resistance: A cautionary note. In *Proceedings of the Second USENIX Workshop on Electronic Commerce - Volume 2*, 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. `http://goo.gl/7Khrz8`.

[4] Peter Gutmann. Data Remanence in Semiconductor Devices. In *10th Conference on USENIX Security Symposium - Volume 10*. USENIX Association, 2001.

[5] Philipp Hof. A Primer To Microcontrollers. `http://goo.gl/Pvfb2r`, 2010.

[6] Andrew Huang. Hacking the PIC 18F1320. `http://goo.gl/Xv7nkS`, 2007.

[7] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. In *19th Annual International Cryptology Conference on Advances in Cryptology*. Springer-Verlag, 1999.

[8] Sergei P. Skorobogatov. Semi-invasive attacks – A new approach to hardware security analysis. Technical report, University of Cambridge, Computer Laboratory, 2005.

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