Microcontroller Based IoT System Firmware Security: Case Studies

Chao Gao*,Lan Luo†, Yue Zhang‡, Bryan Pearson†, Xinwen Fu†*
*Department of Computer Science, University of Massachusetts Lowell, MA, USA. Email: cgao@cs.uml.edu
†Department of Computer Science, University of Central Florida, FL, USA.
Email: {lukachan,bpearson}@knights.ucf.edu, xinwenfu@ucf.edu
‡Department of Computer Science, Jinan University, Guangzhou, China. Email: zyueinfosec@gmail.com

***Abstract*—The Internet of Things (IoT) has attracted much interest recently from the industry given its flexibility, convenience and smartness. However, security issues are amongst the most colossal concerns for IoT. This paper studies the security of microcontroller (MCU) based IoT firmware. Given the varieties of MCUs and their running environments, we perform case studies to explore flaws and defense measures of the firmware upgrade strategies. Specifically, we present two attacks, physical attack through the UART port and remote attack through the vulnerable Over-the-Air programming (OTA) update process, against popular air quality monitoring devices from PurpleAir. We investigate a prototype of a secure firmware upgrade system on an ATmega1284P chip, and discuss potential pitfalls identified through our own practice. These pitfalls may occur in the implementation of firmware upgrade by other manufacturers. To the best of our knowledge, we are the first to investigate hardware security of air quality monitoring sensor networks.**

***Index Terms*—MCU, Firmware Update, Security, OTA.**

I. Introduction

The Internet of Things (IoT) is a novel paradigm that has received a lot of attention, featuring a wide range of applications, such as smart home, smart city, smart healthcare and smart environment. For example, purpleAir.com offers a popular low-cost air quality sensing service through a web portal with Google Map, and has one of the largest operational low-cost sensor networks worldwide. The network is being used by individuals and non-profit and governmental agencies for community air quality monitoring purposes (e.g. Governmental agencies in California, Oregon and Washington; non-profit in North Carolina, individuals in Utah and Illinois). IoT offers unprecedented convenience to users. The market of IoT is booming and estimated to reach 25 billion by 2021 according to Gartner [1], and Forbes [2] research also shows that more than 51% of companies have launched IoT projects.

However, one of the biggest concerns is the rampant security issues in IoT products, which hinder the development of IoT market. According to SonicWall, there were 32.7 million attacks against IoT in 2018, while these attacks have been escalating throughout 2019 with an explosive growth [3]. The attack surface of IoT involves various dimensions, including hardware, operating system/fimrware, software, networking and data. Among them, the firmware attacks are particularly challenging to address due to its heterogeneousness. Even for a specific type of attack, such as a buffer overflow attack, there is no general countermeasure that can be adopted to work unanimously across different platforms. This paper studies attacks that may occur during a firmware update of MCU based IoT systems and discusses potential defense measures. Specifically, instead of proposing a general firmware update attack or defense measure, which may not be realistic given the variety of the hardware, we conduct case studies to gain real world insight into these security issues. The motivation of this paper is to offer guidelines for implementing secure firmware updates, and to identify potential pitfalls during implementation by IoT vendors.

We first demonstrate how an attacker can break IoT security via a flawed firmware update mechanism. In this case study, we investigate popular air quality monitoring devices from PurpleAir [4]. For clarity, we use the wording “device” to refer to the PurpleAir “sensor”, to different the air quality monitoring device from its actual particulate matter (PM) sensor PMS5003 [5] inside the device. In [6], we have successfully exploited the communication protocol of the device, breaking its data integrity. In this paper, we exploit the firmware of the device. We systematically analyze the firmware update mechanism and find that the firmware update mechanism does not involve any authentication or encryption strategies. We therefore deploy two types of attacks, a physical hardware attack and a remote attack. In the hardware attack, we assume that the attacker can physically access the air quality device and set up a connection to the MCU via its debugging UART (Universal Asynchronous Receiver/Transmitter) interface. This is a reasonable assumption since air quality monitoring devices can be placed outside and free to touch by anyone. The attacker can then either flash a malicious firmware into the device or steal the sensitive information such as WiFi passcode from the firmware. In the remote attack, the attacker can impersonate as a server, fabricating a malicious firmware and sending it to the device, which accepts the malicious firmware without no verification.

Efforts have been taken to counter attacks [7], [8] against MCU. MITRE Cyber Academy [9] proposes requirements for designing a secure firmware update system for a legacy MCU ATmega1284P, an AVR-architecture based chip, which does not have WiFi or other networking capabilities. The firmware shall be encrypted and distributed over the Internet. Authorized parties with right credentials can update the firmware. ATmega1284P features internal security features such as a lock-bit mechanism to prevent hardware based I/O attacks. Our implemented secure firmware update system contains four components, including a secure bootloader, a firmware protection tool, a firmware update tool, and a readback tool. We assume that an AES key is predefined and known only by the vendor tools and secure bootloader. The secure bootloader is used to verify the integrity of the firmware and perform decryption with the predefined key. The secure bootloader also installs the firmware into the chip. The firmware protection tool is used to encrypt the firmware when the firmware is released over the Internet. The firmware update tool works with the secure bootloader cooperatively to install the firmware. The readback tool is designed for vendors to extract the firmware out of the chip when unexpected errors occur on the chip after deployment so that the vendors can perform firmware analysis to diagnose the issues. Although the firmware update tool is design for legacy MCUs with no Internet capability, the strategy can be easily extended to MCUs with Internet capability with similar modules.

We discuss possible pitfalls that may occur during the implementation of a firmware update system. We identify these pitfalls through our own practice, and believe that these pitfalls may occur during the implementation of other applications. For example, misconfiguration of lock bits allows read and write of the firemware through JTAG and SPI (Serial Peripheral Interface). The clock glitch attack, which is caused by overclocking [10] a microcontroller (MCU), may bypass the encryption procedures in code. A key loss will allow arbitrary change to the firmware. While we are aware that the attacks and pitfalls we demonstrate are just the tip of the iceberg, we hope that these issues can educate vendors and help to avoid similar pitfalls.

Our major contributions can be summarized as follows:

1) We systematically analyze the firmware update mechanisms of popular PurpleAir air quality monitoring devices and present hardware and remote attacks against these devices. We are the first to launch a firmware exploit against air quality monitoring devices.
2) We design and implement a secure firmware update system for a popular legacy MCU ATmega1284P according to the requirements from MITRE Cyber Academy, addressing the attacks that may occur during the firmware update. We provide guidelines on how to design and implement firmware update security for such legacy MCUs.
3) We present a variety of potential pitfalls of the secure firmware update system based on our practice. Our goal is to help vendors to avoid these pitfalls in their own implementation.

Responsible disclosure: We have notified Purpleair our findings. They are working with us cooperatively addressing air quality monitoring sensor security.

Roadmap: The rest of this paper is organized as follows. In Section II, we briefly introduce the primitives involved in this paper, including MCU, Over-The-Air (OTA) and so forth. In Section III, we present the attacks against air quality devices from PrepleAir. We introduce the detailed design of a secure firmware update system in Section IV. Related work is presented in Section V and we conclude the paper in Section VI.

II. Background

In this section, first present a brief introduction to microcontrollers (MCUs), and then show how MCUs obtain Internet connectivity. Finally we introduce the Over-The-Air (OTA) firmware update mechanism.

A. Microcontrollers

A MCU is a small computer on a single chip designed for embedded systems. Compared with other programmable chips, MCUs are often characterized with low power consumption, lower price, small size, but limited computation and memory resources. MCU can be extended with additional functional modules such as external flash and various sensors. These features lead to a broad adoption of MCUs for lightweight IoT applications such as air quality sensor networks.

An MCU chip often consists of a central processing unit (CPU), system clock, memory, and peripherals [11]. The firmware, also called the application image, can be burned to the memory (internal or external) of the MCU and executed. Due to its restricted resources and relatively simple architecture, MCUs are often dedicated to one or more simple tasks instead of processing multiple complex tasks simultaneously.

B. Internet Connection

In the context of IoT, the wireless Internet connection is often an essential element for connecting the small gadgets to much more powerful servers. The wireless Internet interfaces of MCU can be categorized as internal or external based on their integration methods. The module could also be further categorized based on their functionality features. Some manufacturers have integrated a network module into the MCU as its basic component. These integrated interfaces can WiFi or Bluetooth low energy (BLE), which are the most commonly used interfaces in IoT gadgets [12]. MCU can be equipped with integrated Ethernet and wired connectivity options and often found in industry applications.

Even though modern MCUs often have integrated network interfaces, a considerable number of MCUs, particularly legacy ones, leave Internet connection as optional. Owing to the extensibility of MCUs, a developer can still apply such MCUs to the IoT gadgets by adding their choice of external network interfaces. For instance, the ATWINC1500 add-on WiFi module from Microchip provides a WiFi connection to the microcontroller and the WL1831MOD module from Texas Instrument offers both WiFi and Bluetooth connections.

In recent years, a new type of connectivity technology, cellular low-power wide-area network (cellular LPWAN), came into IoT applications. This technology provides a low-cost solution to the Internet connection of IoT systems, particularly fit for application sending small amounts of data through the network. Based on the traditional Long-Term Evolution (LTE) standard for mobile device wireless communication, Narrowband IoT (NB-IoT) and Cat-M1 are two typical types of cellular technology for IoT devices. The service of NB-IoT and Cat-M1 are provided by the cellular providers such as AT&T and Verizon within the U.S.. By connecting to the nearby cellular towers of the service provider, IoT devices can connect to the network using different bands from the standard LTE. The cellular IoT modules that provide NB-IoT and Cat-M1 can also be connected to the MCU externally to provide Internet connection.

MCUs are often equipped with protocols such as Universal Asynchronous Receiver/Transmitter (UART) and Serial Peripheral Interface (SPI), providing convenient connections between the external Internet modules and MCUs.

C. Over-The-Air mechanism

OTA is a mechanism that devices can use to wirelessly download a newly released firmware from remote servers and reprogram itself. Given the potential large scale of an IoT system, the OTA mechanism is well suited for firmware update. Upgrading through secure OTA is critical for improving the lifetime of these smart gadgets [13]. Not only does OTA increase the functionality and scalability of the devices, but security vulnerabilities can be fixed even after the devices have been launched. Costs for maintaining IoT applications are largely reduced by automatically updating remotely.

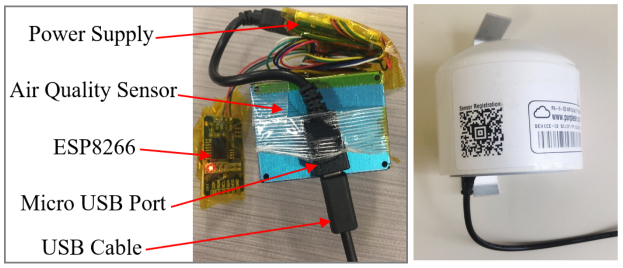
However, network vulnerabilities may exist in the data downloading process if the network protocol, e.g., WiFi or BLE, is not secured. For instance, an OTA task built atop the Hypertext Transfer Protocol (HTTP) sends the firmware to the IoT devices without a firmware authentication process or data encryption mechanism. Malware may get into the IoT device through the OTA process. Therefore, to ensure the scalability and security of the IoT devices, a secure OTA mechanism has to be implemented.

III. Case Study - PurpleAir

In this section, we first give a brief introduction to the PurpleAir air quality monitoring device, and then present the analysis of the firmware, revealing its layout. Finally, we discuss the potential exploits against the device.

A. PurpleAir Air Quality Monitoring Devices

Fig. 1 (a) shows commercial Purpleair device. We removed its case and Fig. 1 (b) shows its internal components, including two air quality sensors PMS5003, an MCU, and a power supply circuit board. The air quality sensor measures ambient mass (e.g. particle number concentrations of PM2.5, humidity) and is connected to the MCU through UART. The MCU is an ESP8266 [14] chip and reads the measurements from the air quality sensor through UART. ESP8266 features various functionalities such as WiFi connectivity and peripheral ports. The power supply circuit board is used to power all components. Collectively, these components can monitor multiple environmental metrics and report the measurements to the cloud server. PurpleAir offers an OTA mechanism, allowing the device to download a newly released firmware from the cloud server.



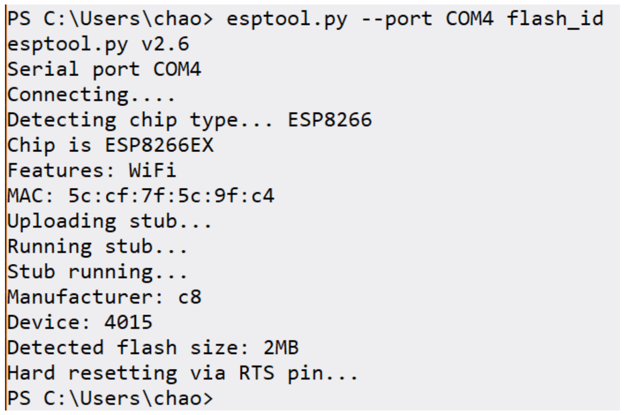
(a) PurpleAir Device Without Case (b) PurpleAir Device with Case

Fig. 1. PurpleAir Device

B. Access to Firmware

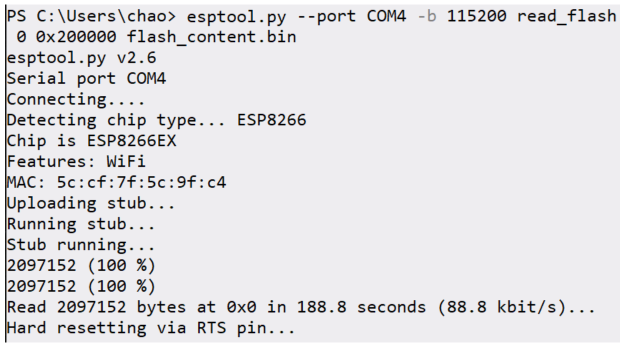
We now present the hardware attack to access the flash of the device, which uses the ESP8266 MCU. The Purpleair device comes with a "charge-only" Micro-USB cable without data wires. However, a generic smartphone Micro-USB cable enables the programming functionality through the cable. The ESP8266 offers a universal asynchronous receiver-transmitter (UART) port for data transmission, e.g. programming. To enable the UART communication, we connect the device to a debugging computer via a Micro-USB cable with a baud rate of 115200. We use a Python based tool esptool [15] at our testing computer to receive the data sent from the device. Esptool is designed to communicate with the ROM bootloader of Espressif Systems [16]. An example of the Espressif System is the ESP family of MCUs, such as ESP8266 and ESP32 [17]. Esptool offers various functionalities. We list a few of these functionalities that are closely related to our research.

1) "esptool.py -port *PORT* flash_id". As shown in Fig. 2, this command can read the basic information of a firmware, such as the device MAC address, flash size and manufacturer information. The PORT argument refers to the UART port number.
2) "esptool.py -port *PORT* -b 115200 read_flash *0 0x200000 flash_contents.bin*". As shown in Fig. 3, this command reads the flash content out of the chip. In our case, esptool reads $0x200000$ (≈2M) bytes starting from address 0 of the flash memory and saves it onto the local disk with a file name *"flash_contents.bin"*.
3) "esptool.py erase_flash". This command erases All bytes of the flash will be replaced by meaningless "0xFF" bytes. Similarly, "erase_region" erases a specific data section of the flash memory, with specified parameters including the starting address and flash size as shown in previous commands.
4) "esptool.py -p *PORT* write_flash *0x1000 my_app.bin*". This command flashes a binary file (*my_app.bin* in our



```
PS C:\Users\chao> esptool.py --port COM4 flash_id
esptool.py v2.6
Serial port COM4
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
MAC: 5c:cf:7f:5c:9f:c4
Uploading stub...
Running stub...
Stub running...
Manufacturer: c8
Device: 4015
Detected flash size: 2MB
Hard resetting via RTS pin...
PS C:\Users\chao>
```

Fig. 2. Reading basic information of the ESP8266.



```
PS C:\Users\chao> esptool.py --port COM4 -b 115200 read_flash
 0 0x200000 flash_content.bin
esptool.py v2.6
Serial port COM4
Connecting....
Detecting chip type... ESP8266
Chip is ESP8266EX
Features: WiFi
MAC: 5c:cf:7f:5c:9f:c4
Uploading stub...
Running stub...
Stub running...
2097152 (100 %)
2097152 (100 %)
Read 2097152 bytes at 0x0 in 188.8 seconds (88.8 kbit/s)...
Hard resetting via RTS pin...
```

Fig. 3. Reading flash contents from the ESP8266.

case) onto the chip. The parameters are similar to the previous commands.

C. PurpleAir Flash Map

Based on the analysis above, we now present the flash map from the PurpleAir device. The size of the flash is 2MB, while the last 1MB is meaningless padding data. The ESP8266's on die Read-Only Memory (ROM) contains some library code and a first stage boot loader [18], [19]. The PurpleAir device supports OTA and its flash is split into four sections, including two program images, an Electrically Erasable Programmable Read Only Memory (EEPROM) region, and a default data section. As shown in Fig. 4, each program image contains two sections, denoted as *boot.bin* and *user.bin*. *boot.bin* is used to store the bootloader. *user.bin* is used to store the application code. The two program images can be identical. EEPROM is a type of non-volatile memory that provides persistent storage of data across reboots. In the case of PurpleAir, the WiFi SSID and password are stored in this region. Finally, the default data section contains the images of *esp_init_data_default.bin* file and *blank.bin*, which store default system parameters, such as Wi-Fi configurations other than SSID and password [20]. The flash map of the PurpleAir device is customized and different from the official OTA implementation of ESP8266 [21]. For example, the official OTA implementation has only one *boot.bin*, which stores the address of the active program image and runs that program at restart.

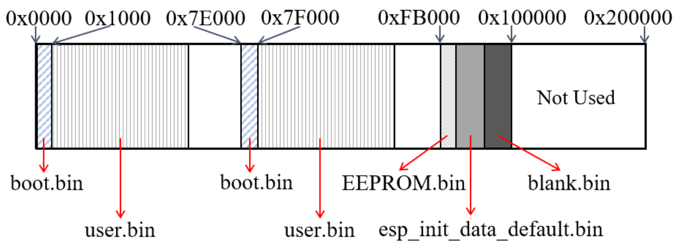


Fig. 4. PurpleAir Firmware Flash Map

D. Exploits and Attacks

The ultimate goal of our attack is to change the firmware of the device. To this end, two types of attacks will be presented, namely the physical attack and remote attack. In the physical attack, we assume that the adversary can physically access the device. This assumption is reasonable since the air quality device can be placed outside and free to access. Furthermore, the device may be left unmonitored for an extended period of time. However, in the remote attack, we assume that physical access is unavailable to the adversary.

Physical attack: An adversary can deploy the physical attack by connecting the MCU to a computer through a Micro-USB cable. We list some potential exploits that can arise from this attack to demonstrate its grave consequences : (i) *Flashing a malicious firmware*. An attacker can program a malicious firmware and flash it to the chip. The modified firmware can inject fabricated data into the air quality sensor network, which may misinform the public. (ii) *Stealing Wi-Fi credentials.* Due to the lack of authentication and encryption of the firmware, the flash contents are free to access. We can therefore extract the Wi-Fi credentials out from the flash, which causes damages beyond the air quality sensor network.

Remote attack: Recall that the PurpleAir offers an OTA mechanism, which allows the device to update its firmware after deployment. Through traffic analysis, we know that when a PurpleAir device reboots and connects to the cloud server, the device sends a request to the cloud server and queries the current firmware version. If there is a new firmware, OTA will be performed to synchronize the firmware with the cloud server. This mechanism ensures the device is up-to-date. After carefully analyzing the PurpleAir OTA mechanism, we identify a severe vulnerability: the device does not authenticate the server, and the server does not authenticate the device. That is, there is no mutual authentication between the device and server. Moreover, the communication is in plaintext. The firmware is not protected with any mechanism and in plaintext.

The lack of mutual authentication raises security issues. As shown in Fig. 5, an attacker can either pretend to be the cloud server or a rogue device to cause mischief: (i) If the attacker is able to impersonate the server, the attacker can trivially fabricate a malicious firmware and send it to the device, which updates the local firmware with the malicious

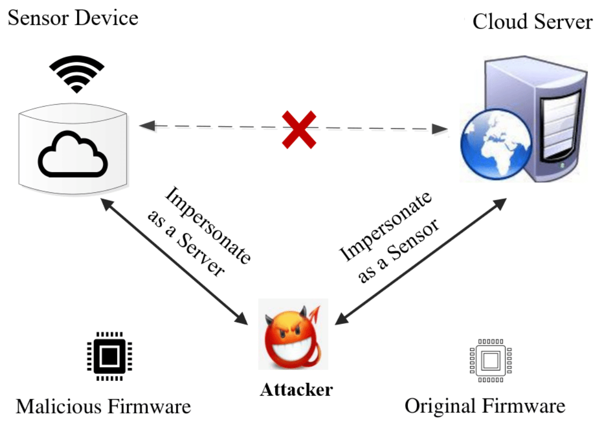
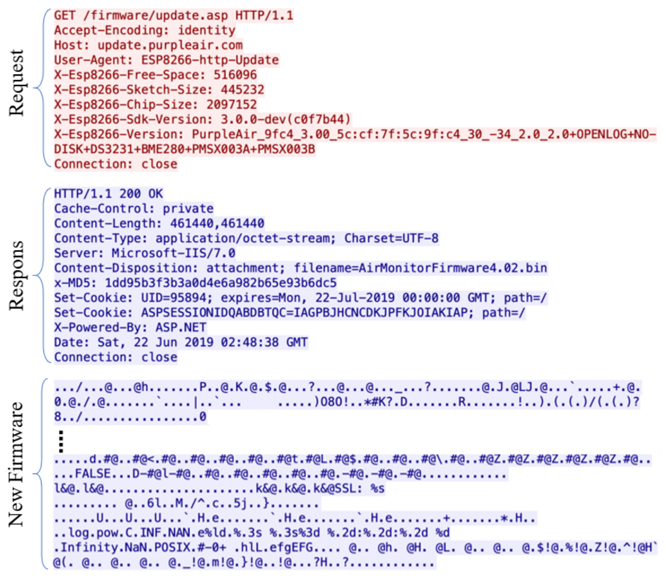


Fig. 5. The overview of remote attack

firmware with no verification process. The malicious firmware will execute after reboot. Here is one way to impersonate the server. The attacker can jam the WiFi router that the victim device is connected with. After a period of connection failures, the PurpleAir device automatically resets itself to the AP mode, which allows the attacker to configure the victim device, which will connect to a rogue WiFi router. The rogue WiFi router can then impersonate the server. (ii) When the attacker impersonates a device, the attacker can obtain the firmware at the server by sending a query to the cloud server, as shown in Fig 6. We have confirmed these two attacks in our experiments.



```
GET /firmware/update.asp HTTP/1.1
Accept-Encoding: identity
Host: update.purpleair.com
User-Agent: ESP8266-http-Update
X-Esp8266-Free-Space: 516096
X-Esp8266-Sketch-Size: 445232
X-Esp8266-Chip-Size: 2097152
X-Esp8266-Sdk-Version: 3.0.0-dev(c0f7b44)
X-Esp8266-Version: PurpleAir_9fc4_3.00_5c:cf:7f:5c:9f:c4_30_-34_2.0_2.0+OPENLOG+NO-
DISK+DS3231+BME280+PMSX003A+PMSX003B
Connection: close

HTTP/1.1 200 OK
Cache-Control: private
Content-Length: 461440,461440
Content-Type: application/octet-stream; Charset=UTF-8
Server: Microsoft-IIS/7.0
Content-Disposition: attachment; filename=AirMonitorFirmware4.02.bin
x-MD5: 1dd95b3f3b3a0d4e6a982b65e93b6dc5
Set-Cookie: UID=95894; expires=Mon, 22-Jul-2019 00:00:00 GMT; path=/
Set-Cookie: ASPSESSIONIDQABDBTQC=IAGPBJHCNCDKJPFKJOIAKIAP; path=/
X-Powered-By: ASP.NET
Date: Sat, 22 Jun 2019 02:48:38 GMT
Connection: close

.../...@...@h.......P..@.K.@.$.@...?...@...@..._...?.......@.J.@LJ.@...`.....+.@.
0.@./.@.......`....|..`...    .....)08O!..*#K?.D.......R.......!..).(.(.)/(.(.)?
8../................0
⋮
.....d.#@..#@<.#@..#@..#@..#@..#@t.#@L.#@$.#@..#@..#@\.#@..#@Z.#@Z.#@Z.#@Z.#@Z.#@..
...FALSE...D-#@l-#@..#@..#@..#@..#@..#@.-#@.-#@.-#@...........
l&@.l&@....................k&@.k&@.k&@SSL: %s
......... @..6l..M./^.c..5j..}.......
......U...U...U...`.H.e.......`.H.e.......`.H.e.......+.......*.H..
..log.pow.C.INF.NAN.e%ld.%.3s %.3s%3d %.2d:%.2d:%.2d %d
.Infinity.NaN.POSIX.#-0+ .hlL.efgEFG.... @.. @h. @H. @L. @.. @.. @.$!@.%!@.Z!@.^!@H`
@(. @.. @.. @.. @._!@.m!@.}!@..!@...?H..?...........
```

Fig. 6. PurpleAir update response packet with new firmware attached.

IV. Secure Firmware Update

To counter attacks against the firmware update, many efforts have been taken [7], [8], [22]. One major issue of the PurpleAir device is that it uses the chip ESP8266, which does not have any built-in hardware mechanism for security. For example, there is no way to disable its UART port. An open UART port is subject to the physical attack in Section III. A secure IoT device needs hardware security features. To defeat remote attacks against the firmware update, mutual authentication and encryption shall be used. In this regard, MITRE Cyber Academy [9] proposes requirements for designing a secure firmware update system for legacy chips with no Internet connection so that secure firmware can be downloaded from the Internet and updated locally. We design and implement such a secure firmware update system according to the proposed requirements and discuss the potential pitfalls.

A. Overview

To guarantee a secure firmware update, we shall incorporate encryption and authentication during the update process. Specifically, the vendor shall encrypt the firmware before release, while the decryption key has to be secure inside each product. Only the product can decrypt the firmware. The product has to verify the integrity of the firmware too. To this end, the hash value of the target firmware is encrypted and attached together with the encrypted firmware. Finally, to prevent the attacker from leveraging physical attacks and obtaining sensitive data, such as the decryption key or the firmware, the debug interfaces, such as JTAG, SPI and UART ports, have to be disabled or at least have limited accessibility (e.g. protected by a strong password). The lock bits should also be enabled if the chip of the product supports it. In this way, firmware analysis will fail.

The secure firmware update system as shown in Fig. 7 consists of four components, including a secure bootloader, a firmware protection tool, a firmware update tool, and a manufacturer readback tool. A typical use of the secure firmware update system works as follows. The vendor uses the firmware protection tool to encrypt the firmware with an AES key and also uses the programmer board (AVR Dragon [23]) in Fig. 8 to flash the secure bootloader into the chip's bootloader flash section through the programmer board's *ISP* interface, which is connected to a set of programmable I/O pins on the ATmega1284P. The vendor's AES key is hard coded into the bootloader. During a firmware update process by a vendor or a user, the firmware update tool sends the encrypted firmware to the secure bootloader through the UART port. The bootloader decrypts and verifies the firmware, and copies the decrypted firmware into the application flash memory section. The device then reboots to run the new firmware. The vendor can use the readback tool to extract the (damaged) firmware from the chip for debugging through the UART port. During the readback, the bootloader encrypts the firmware and the vendor has to decrypt the encrypted firmware.

In the real world scenario, the firmware update tool can be replaced with the internal OTA mechanism introduced in Section II. In this case, the bootloader obtains a firmware from the cloud server, then flashes it into the chip. The other functions of the bootloader and the other three tools will work without any changes. However, in some cases, the firmware update tool may be preferable—for instance, when the new

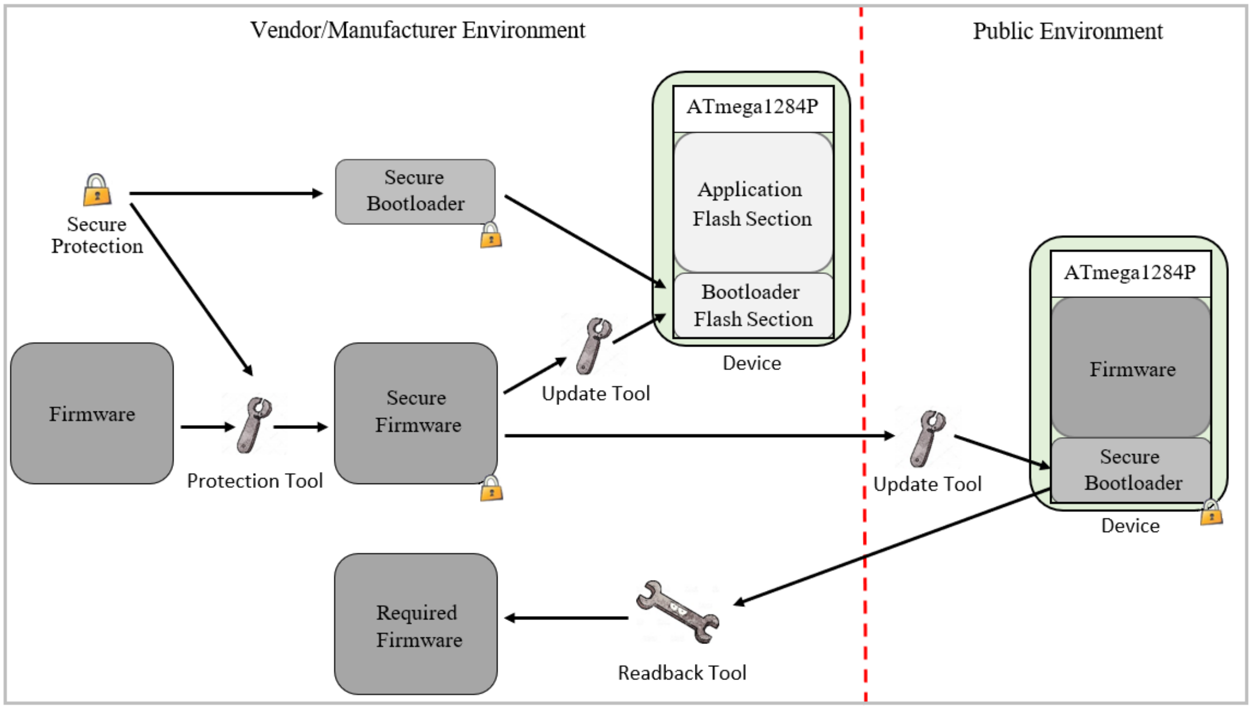


Fig. 7. An overview of the secure firmware update System

firmware is too large to download without overwriting the previous firmware. For the sake of brevity, we will not reiterate details on MCU-based OTA, which were already discussed in Section II.

B. Detailed Design

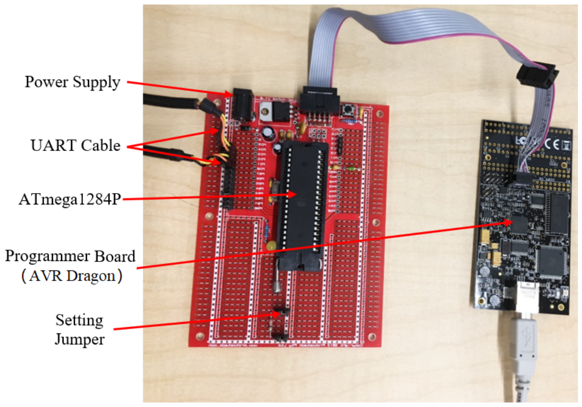


Fig. 8. A secure firmware update system

We now present a detailed design of the secure firmware update system. As a demonstration, we implement the design on an ATmega1284P chip shown in Fig. 8. ATmega1284P is a high performance, low power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture, featuring 128KB ISP flash memory, 16KB SRAM, 4KB EEPROM, 32 General Purpose Working Registers, two UART interfaces, an SPI serial port, and a Joint Test Action Group (JTAG) test interface. JTAG is provided for on-chip debugging and programming. In regards to security, the ATmega1284P provides a program locking mechanisms for software security. Specifically, the MCU has three fuse bytes, i.e. low fuse byte, high fuse byte, and extended fuse byte, and a lock byte. The low fuse byte is used to deal with the clock related operations. The high fuse byte has several different settings, such as enabling the JTAG/SPI programming port, watchdog timer and bootloader size settings. The extended fuse byte is used to set the brown-out detection trigger level, which refers to the voltage level setting. The lock byte is employed to control the read and write permissions between the bootloader section and the application flash section. It can be used with the high fuse byte to prevent unauthorized read and write access to the flash memory and EEPROM through JTAG and SPI.

The ATmega1284P flash memory is divided into two sections: the application section and the bootloader section. The high byte fuse is used to allocate the size of these two sections. The bootloader section is typically used to write the application firmware to the application flash section. To modify the bootloader, we need an external programmer board (such as the one shown in Fig. 8) to write the bootloader to the MCU. The application firmware may use the code built into the bootloader [24]. Overwriting the bootloader section requires the use of Store Program Memory (SPM) instructions [25]. To prevent the application firmware from destroying the bootloader, SPM instructions are not allowed to be executed in the address space of the application flash section. However, it is shown [26] that the application has a trick to modify the bootloader on the AVR microcontroller by setting an interrupt in the application firmware and using the "jmp" instruction to force the AVR to jump to the bootloader section. The bootloader can then be overwritten by the existing SPM instructions inside the bootloader section.

*1) **Secure Bootloader:*** In our current implementation, the secure bootloader can work in three modes: firmware loading mode, firmware booting mode, and readback mode. The bootloader is programmed to read the jumper setting in Fig. 8 to get into different modes. During the firmware loading mode, the bootloader receives an encrypted firmware from the firmware update tool, and verifies its integrity, which prevents arbitrary modification of the encrypted firmware. Once verification succeeds, the secure bootloader decrypts the firmware and copies it into the application flash memory section. Afterward, the bootloader enters the firmware booting mode and boots the image in the application flash memory. The bootloader may also be in the readback mode. When the bootloader is in this mode, the vendor can communicate with it and the bootloader will copy and encrypt the firmware out of the chip. This may occur when the chip has some unexpected errors and a firmware analysis must be done for diagnosis. In this case, the bootloader requires the readback tool to provide a username and password for authentication, which avoids unauthorized access. These credentials were predefined and stored in the EEPROM by the vendor. Since we enable the hardware-based lock bits of the ATmega1248P, an attacker cannot compromise the firmware or the sensitive credentials via physical attacks.

*2) **Firmware Protection Tool:*** The firmware protection tool is used to generate a secure firmware, which contains five phases : (i) We generate an AES encryption key and configure the firmware protection tool to have access to the key. Before deployment, the AES key shall also be stored in the chip where the firmware will be installed. This can be done by installing an initial version of the firmware, where the key is hard-coded. (ii) The firmware protection tool encrypts the intended firmware with the AES key generated above. (iii) The firmware protection tool feeds the encrypted firmware into a hash function and computes the hash value. We denote the hash value as the ID of the firmware, since it uniquely refers to the firmware. (iv) The firmware protection tool feeds the firmware ID and other basic information of the firmware (e.g. the size, version number) into the hash function. The generated hash value here is used as the firmware checksum for the verification step. We then encrypt the hash checksum with the pre-defined key. The encrypted checksum will be a part of the secure firmware, so that the integrity of the firmware is also guaranteed. (v) The firmware protection tool uses the previous steps to generate a secure firmware. The secure firmware can be split into three parts as shown in Fig. 9: the *header*, the *encrypted firmware* and a *release message*. The header contains the firmware size, firmware version number, firmware ID, encrypted checksum and will be stored in the EEPROM. The release message provides a basic description of the secure firmware and will be written into the application flash memory section following the firmware. When the device boots, the bootloader can write the release message to UART to display to the user.

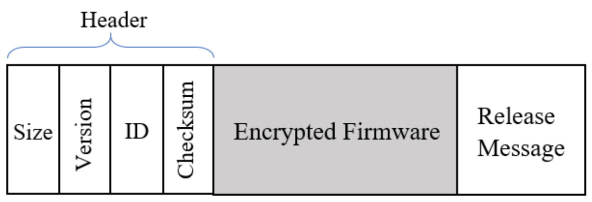


Fig. 9. Secure firmware structure

*3) **Firmware Update Tool** :* The firmware update tool assists the bootloader by copying the firmware into the application flash memory, which contains two phases: (i) The firmware update tool sends the header of the secure firmware to the secure bootloader and waits for the verification response from the bootloader. In this phase, the bootloader is in the firmware loading mode and will perform the basic verification of the firmware, as discussed earlier. (ii) When the verification passes, the firmware update tool sends the encrypted firmware to the bootloader and waits for the verification response from the bootloader. After the bootloader successfully receives the encrypted firmware and verifies its integrity, the firmware update tool terminates. The bootloader decrypts the firmware and writes the firmware into the application flash memory section.

*4) **Firmware Readback Tool:*** The readback tool communicates with the secure firmware via the UART port after deployment, allowing the vendors to obtain an encrypted firmware for analysis. To avoid unauthorized access to firmware, the secure bootloader will require a username and password which were predefined by the vendor. Moreover, all data transferred between the readback tool and bootloader is encrypted by the predefined AES key.

C. Potential Pitfalls

*1) **Misconfiguration:*** Misconfiguration of the chip may lead to severe consequences. A common misconfiguration is the failure to enable the fuse or lock bits, which prevents hardware-based cracking after deployment.

Enabling the fuse bytes can prevent an attacker from accessing the memory via the JTAG/SPI port. Configuring the lock bits can prevent an attacker from programming the flash and EEPROM by utilizing the High Voltage Parallel Programming (HVPP). The AVR Dragon programmer board can communicate with the ATmega1284P in the HVPP mode by connecting its *HV_PROG* interface to multiple programmable I/O pins on the chip. With HVPP, if the lock bits are not configured and fuse bytes are properly set, attackers can use the Atmel Studio software [27] to rewrite the fuse bytes through the programmer AVR Dragon board to enable the JTAG/SPI ports and read the firmware and EEPROM memory through these ports. Recall that sensitive information is stored in EEPROM memory, such as decryption keys, credentials and firmware information. Misconfiguration may allow an attacker to obtain sensitive information and modify it. In fact, even with enabled fuse bytes and lock bits, with the programmer board, an attacker can rewrite the entire firmware and EEPROM while she/she cannot read them. Therefore, ATmega1284P may not be an ideal IoT chip if rewriting the flash is a concern.

*2) **Clock Glitch Attack**:* For most integrated circuits (ICs), such as an MCU in our case, clock signals are used to synchronize the various internal components in an integrated circuit. The system clock frequency provided by the chip manufacturer is the frequency at which the clock signal reaches each IC component correctly, and all instructions can be executed normally at this frequency.

Overclocking is a practice of increasing the system clock frequency beyond the frequency provided by the manufacturer. As a result, overclocking can improve the CPU/MCU performance to some extent. However, this is not universally true due to hardware limitations. Extra heat may be generated in this case, which may hinder the performance as well. Therefore, an IC may not work properly when overclocking occurs. A clock glitch is a short period of time when the system clock frequency suddenly increases. It may lead to incorrect instruction execution and unstable data output. An attacker can perform the clock glitch attack to bypass cryptographic instructions such as encryption and integrity check.

*3) **The Leakage of the Key**:* An attacker may obtain the encryption key in various ways due to the carelessness of vendors or the flaws existing in tools. For example, the key can be hard-coded in the firmware protection tool. In this case, an attacker can perform the reverse engineering process and extract the key from the binary file. Even if the key is hard-coded in the encrypted firmware, another potential pitfall involves using an insecure encryption algorithm, which may be subject to cryptanalysis. Under such circumstances, an attacker can break the encryption of the firmware and obtain the hard-coded key that is stored in the EEPROM. Once the attacker obtains the key, various other attacks can be deployed, such as fabricating a malicious firmware. As a solution, the vendors should keep their encryption keys in a safe location.

*4) **Bypassing the Firmware Verification**:* We now demonstrate a flawed design of the firmware update process. The firmware update tool loads the firmware page by page into the chip. That is, the chip first receives the data from the update tool and caches it into a buffer. The buffer will hold the data until it reaches the size of one page, i.e., 256 bytes. The bootloader decrypts this page and write it into the application flash memory section of the chip. This process repeats until the entire firmware has been written to the chip. The firmware update tool then sends a termination message to the chip. Firmware integrity verification is then performed. If the verification fails, the bootloader erases the application flash memory given that the damaged/malicious firmware is already written into the application flash memory section.

This firmware update strategy above is flawed. If an attacker obtains the firmware update tool and tries to attack a device, he can change the tool so that a junk firmware is uploaded into the device, but the crooked firmware update tool does not send the termination message so that the verification is not performed. This will bypass the verification process and a junk firmware is written into the device.

One patch to defeat this type of attack is to divide the entire application flash program memory into two partitions: the first partition for the current firmware and the second partition for the new firmware. The update tool loads the new firmware in the second partition and verifies its integrity. After the integrity is verified, the new firmware is copied to the first partition. An over-the-air update (OTA) often uses this strategy of multiple partitions in the firmware update process although the actual OTA process can be different.

*5) **Flaws Existing in Readback Tool**:* Recall that the bootloader requires user credentials (i.e username and password) when the vendor tries to use the readback tool to communicate with it. If the vendor remotely interacts with the bootloader through the Internet and the communication channel is not secured (e.g., no encryption), an adversary may intercept the communication content, including the user credentials. The adversary can use the credentials to read or modify the firmware. Therefore, the readback tool must enforce proper network security practices.

D. Discussion

Table I summarizes how to protect the I/O interfaces of ATmega1284P. ATmega1284P cannot disable UART, which shall be protected by a secure bootloader. The bootloader can be programmed and control the interaction with the chip through UART. As discussed above, ATmega1284P is actually not an ideal chip for secure IoT applications. With the programmer board in Fig. 8, the bootloader and firmware of ATmega1284P can be overwritten arbitrarily even with enabled fuse bytes and lock bits.

TABLE I
ATMEGA1284P I/O SECURITY MEASURES

I/O	**Functionality**	**Security Measures for I/O Interfaces**
JTAG	Test interface for on-chip debugging and programming. Flash, EEPROM, fuse and lock bits can be programmed via this interface.	Fuse bytes + lock bits
SPI	Used for serial communication and data exchange with peripherals; Communicate with the programmer board to download the bootloader.	Fuse bytes + lock bits
UART	Used for data communication with external devices; Communicate with the firmware update tool and the firmware readback tool.	Secure bootloader with integrity check and password

We have performed a thorough survey of recent advances of secure MCU chips (such as ESP32 and CC3220SF) [28] which may meet the requirements of a secure IoT application. A chip for a secure IoT application should support security from five aspects: hardware, operating system/firmware, software, networking and data generated and maintained within the system. (i) Hardware security: Hardware security is critical when the adversary can physically access IoT devices. All debugging ports such as UART/JTAG should be disabled in the final products. (ii) Operating system (OS)/firmware security: Mechanisms are needed to prevent the firmware from being changed by physical hacks and malware. The file system, processes, memory activities and network ports should be constantly monitored to detect dynamic malicious activities. (iii) Software security: Secure coding practices are needed to reduce vulnerabilities in IoT applications. (iv) Network security and privacy: An IoT system is a networked system, and the whole system must be secured from end-to-end [29]. (v) Data security: Sensitive data should be encrypted and flash encryption is preferred if the devices are deployed in the wild.

V. Related work

The firmware attacks and the corresponding defense measures have quite deservedly received much recent attention due to the explosive growth of IoT market. In this section, we will review some attacks against a few mainstream chips as well as their countermeasures.

The Harvard architecture device is one of the most popular chips widely used all over the world. It was originally believed to defend against code injection attacks, since the data region and program region are physically separated. However, Francillon et al. [30] demonstrated a remote code injection attack against the Harvard architecture-based wireless sensor network. A fake stack (containing malware) is injected into the victim's data memory, and then a specially-crafted packet is sent to poke the vulnerabilities existing in the bootloader, leading the bootloader to copy the fake stack from data memory to program memory. In such a way, the malware is injected and allows the attacker to gain full control of the victim sensor. More recently, Krzysztof Cabaj et.al [31] showed that the Harvard architecture based MCU is prone to code reuse attacks, which are a variant of the code injection attack. Their main contribution is a demonstration of the attacks on the Arduino family of devices that use the Webduino web server, which is one of the most popular options for IoT devices. To counter code injection attacks running on sensor nodes, Tan [32] proposed and implemented a remote attestation protocol. Compared with other attestation protocols which may be subject to unauthorized alteration, they run their protocol on a tamperresistant Trusted Platform Module (i.e. an Atmel AT97SC3203S chip), where unauthorized alteration will fail. Sergio et al. [33] proposed a software-based defense for Harvard architectures such as AVR MCUs, which thwart code reuse attacks. To this end, they randomize the code memory of MCUs. The main contribution of this work is a software-based implementation rather than hardware, targeting endpoint users rather than manufacturers.

Shoei Nashimoto et al. [34] showed how buffer overflow attacks, together with multiple fault code injection attacks, can control the program flow of MCUs. They demonstrated their attacks on an AVR ATmega163 microcontroller and a 32-bit ARM Cortex-M0+ microcontroller. To counter the control flow attacks, Francillon et al. [35] proposed a control flow enforcement. The control flow attacks often involve the manipulation of the return stack. In this paper, the authors stored the return stack in protected hardware rather than the typical location. They implemented the solution on an AVR MCU and showed that overhead was minimal.

Sergio et al. [36] implemented a proof-of-concept malware named ArduWorm that can compromise Arduino Yun, a popular IoT platform, by exploiting a memory corruption vulnerability existing in the victims bridge library. Their insight is that the current design of the bridge library lacks proper access control and authentication. The exploit uses code reuse attacks, allowing malware to establish a backdoor and spread through neighbor nodes. The work also proposed possible remedies to mitigate the problem. John et al. [37] addressed two crucial issues in terms of creating stack-safe embedded software: security and efficiency. To address the security problem, they perform a whole program security analysis based on a context-sensitive abstract interpretation of machine codes. To address the efficiency issue, they adopt goal-directed global functions to reduce the stack memory requirements.

We now review related firmware update attacks. Zachry Basnight et al. [38] examine how an attacker can modify the firmware on a Programmable Logic Controller (PLC) [39]. They proposed a firmware analysis methodology to attack such a device. A proof-of-concept attack on an Allen-Bradley ControlLogix L61 PLC was also performed to validate the flexibility of their methodology. However, they do not propose any countermeasures. The effort that is closely related to ours is the research by Ang Cui et al. [40], who also poke the design flaws of the update mechanism of the embedded devices, and demonstrate that an attacker can inject malicious firmware and take control of embedded devices trivially. The examples used in their paper are the Laser-Jet printer. However, the example used in our paper is an air quality sensor. The compromised air quality sensor may inject fake data into the server, which may misinform the public and even mislead policy makers. Additionally, in this paper, the authors discuss possible countermeasures for the update mechanism of the embedded devices, but do not propose the design details and implementation criteria. One major contribution of our paper is that we demonstrate how to implement a defense to firmware update attacks, and we discuss the possible pitfalls. Checkoway et al. [41] show that the CD-based firmware update mechanism of automobiles can also be susceptible to remote compromise, and they offer possible remedies for mitigation. Jong-Hyouk Lee et al. [42] proposed a secure update system based on blockchain [43]. Their observation is that a blockchain system can provide an alternative software solution for data integrity and tamper resistance.

VI. Conclusion

This paper investigates the security issues that may exist in the firmware update process of microcontroller based IoT applications. We demonstrate that the firmware update mechanism of the popular PurpleAir air quality device is subject to both physical hardware attack and remote attack. To counter these attacks, we investigate secure firmware update mechanisms for a legacy mircrontroller ATmega1284P and discuss pitfalls that may occur in implementations. This work and our survey of recent advances of MCUs in [28] will help vendors understand the importance of the secure firmware update mechanism for MCU based IoT systems and avoid potential pitfalls.

Acknowledgement

We thank MITRE Cyber Academy for the ATmega1284P platform and development environment. This work was supported in part by US National Natural Science Foundation grant 1642124. Any opinions, findings, conclusions, and recommendations in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

References

[1] Gartner Inc., "Gartner identifies top 10 strategic iot technologies and trends." https://www.gartner.com/en/newsroom/press-releases/2018-11-07-gartner-identifies-top-10-strategic-iot-technologies-and-trends, November, 2018.

[2] Forbes Inc., "The internet of things: From theory to reality-how companies are leveraging the iot to move their businesses forward." https://www.forbes.com/forbes-insights/our-work/internet-of-things/, 2018.

[3] SonicWall, "Iot attacks escalating with a 217.5% increase in volume." https://www.bleepingcomputer.com/news/security/iot-attacks-escalating-with-a-2175-percent-increase-in-volume/, 2019.

[4] PurpleAir, "Purpleair: Air quality monitoring." https://www2.purpleair.com/, June 11, 2019.

[5] PlanTower., "Pms 5003-pm2.5." http://www.plantower.com/en/content/?108.htmlf, last accessed on Nov. 18, 2019.

[6] L. Luo, Y. Zhang, B. Pearson, Z. Ling, H. Yu, and X. Fu, "On the security and data integrity of low-cost sensor networks for air quality monitoring," *Sensors*, vol. 18, p. 4451, December 16, 2018.

[7] S. Hanna, R. Rolles, A. Molina-Markham, P. Poosankam, J. Blocki, K. Fu, and D. Song, "Take two software updates and see me in the morning: The case for software security evaluations of medical devices.," in *HealthSec*, 2011.

[8] J. Li, C. Chang, D. Shi, W. Xia, and L. Chen, "A new firmware upgrade mechanism designed for software defined radio based system," in *Proceedings of the 2012 International Conference on Information Technology and Software Engineering*, pp. 277–283, Springer, 2013.

[9] M. C. Academy., "ectf: An attack-and-defend exercise for designing secure embedded systems.." https://mitrecyberacademy.org/competitions/ectf-2017/ectf17desc.html, February 28, 2017.

[10] S. Maekinen, "Cpu & gpu overclocking guide," *ATI Technologies, Inc*, pp. 1–26, 2006.

[11] Ravi, "Basics of microcontrollers history, structure and applications." https://www.electronicshub.org/microcontrollers-basics-structure-applications, November 13, 2017.

[12] T. Instruments, "Simplelink wireless microcontrollers." http://www.ti.com/microcontrollers/simplelink-mcus/wireless-mcus/overview.html, 2019.

[13] A. R. Gonzlez, "The importance of ota updates for iot devices." https://barbaraiot.com/articles/importance-ota-updates-iot-devices, January 23, 2019.

[14] Espressif-Systems, "Esp8266: Hardware design guidelines." https://www.espressif.com/sites/default/files/documentation/esp8266_hardware_design_guidelines_en.pdf, 2018.

[15] F. Ahlberg and A. Gratton, "esptool.py." https://github.com/espressif/esptool, April 16, 2019.

[16] E. S. C. Platform, "Esp8266," *Espressif Systems*, 2013.

[17] I. Allafi and T. Iqbal, "Design and implementation of a low cost web server using esp32 for real-time photovoltaic system monitoring," in *2017 IEEE Electrical Power and Energy Conference (EPEC)*, pp. 1–5, IEEE, 2017.

[18] "Esp8266 architecture and arduino gui." https://annefou.github.io/IoT_introduction/02-ESP8266/index.html, Nov 2019.

[19] Espressif-Systems., "Esp8266 sdk: Getting started guide." https://www.espressif.com/sites/default/files/documentation/2a-esp8266-sdk_getting_started_guide_en.pdf, April 16, 2019.

[20] github, "Esp8266: Memory map." https://github.com/esp8266/esp8266-wiki/wiki/Memory-Map, 2018.

[21] E. S. I. Team, "Esp8266 fota introduction." https://www.espressif.com/sites/default/files/99c-esp8266_ota_upgrade_en_v1.6.pdf, 2016.

[22] J. D. Tenbarge, R. Timmerman, M. Nierzwick, R. E. Reinke, D. Birtwhistle, J. R. Long, R. P. Sabo, P. E. Pash, and D. B. Markinsohn, "Firmware update in a medical device with multiple processors," Apr. 19 2012. US Patent App. 12/905,498.

[23] Microchip, "The atmel avr dragon debugger." http://ww1.microchip.com/downloads/en/devicedoc/atmel-42723-avr-dragon_userguide.pdf, 2016.

[24] B. Schick, "Avr bootloader faq." https://www.avrfreaks.net/sites/default/files/bootloader_faq.pdf, May 2009.

[25] Microchip, "Avr assembler instructions - spm." https://www.microchip.com/webdoc/avrassembler/avrassembler.wb_SPM.html, last accessed on Nov. 18, 2019.

[26] Snial, "Bootjacker: The amazing av bootloader hack!." http://oneweekwonder.blogspot.com/2014/07/bootjacker-amazing-avr-bootloader-hack.html, July 2014.

[27] Microchip., "Atmel studio 7." https://www.microchip.com/mplab/avr-support/atmel-studio-7, June 15, 2019.

[28] P. Bryan, L. Lan, Z. Yue, D. Rajib, L. Zhen, B. Mostafa, and F. Xinwen, "On misconception of hardware and cost in iot security and privacy," in *IEEE International Conference on Communications*, 2019.

[29] Z. Ling, K. Liu, Y. Xu, Y. Jin, and X. Fu, "An end-to-end view of iot security and privacy," in *Proceedings of the 60th IEEE Global Communications Conference (Globecom)*, December 2017.

[30] A. Francillon and C. Castelluccia, "Code injection attacks on harvard-architecture devices," in *Proceedings of the 15th ACM conference on Computer and communications security*, pp. 15–26, ACM, 2008.

[31] K. Cabaj, G. Mazur, and M. Nosek, "Compromising an iot device based on harvard architecture microcontroller," in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2018*, vol. 10808, p. 108082G, International Society for Optics and Photonics, 2018.

[32] H. Tan, W. Hu, and S. Jha, "A remote attestation protocol with trusted platform modules (tpms) in wireless sensor networks.," *Security and Communication Networks*, vol. 8, no. 13, pp. 2171–2188, 2015.

[33] S. Pastrana, J. Tapiador, G. Suarez-Tangil, and P. Peris-López, "Avrand: a software-based defense against code reuse attacks for avr embedded devices," in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pp. 58–77, Springer, 2016.

[34] S. Nashimoto, N. Homma, Y.-i. Hayashi, J. Takahashi, H. Fuji, and T. Aoki, "Buffer overflow attack with multiple fault injection and a proven countermeasure," *Journal of Cryptographic Engineering*, vol. 7, no. 1, pp. 35–46, 2017.

[35] A. Francillon, D. Perito, and C. Castelluccia, "Defending embedded systems against control flow attacks," in *Proceedings of the first ACM workshop on Secure execution of untrusted code*, pp. 19–26, ACM, 2009.

[36] S. Pastrana, J. Rodriguez-Canseco, and A. Calleja, "Arduworm: A functional malware targeting arduino devices," *COSEC Computer Security Lab*, 2016.

[37] J. Regehr, A. Reid, and K. Webb, "Eliminating stack overflow by abstract interpretation," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, no. 4, pp. 751–778, 2005.

[38] Z. Basnight, J. Butts, J. Lopez Jr, and T. Dube, "Firmware modification attacks on programmable logic controllers," *International Journal of Critical Infrastructure Protection*, vol. 6, no. 2, pp. 76–84, 2013.

[39] W. Bolton, *Programmable logic controllers*. Newnes, 2015.

[40] A. Cui, M. Costello, and S. Stolfo, "When firmware modifications attack: A case study of embedded exploitation," 2013.

[41] S. Checkoway, D. McCoy, B. Kantor, D. Anderson, H. Shacham, S. Savage, K. Koscher, A. Czeskis, F. Roesner, T. Kohno, *et al.*, "Comprehensive experimental analyses of automotive attack surfaces.," in *USENIX Security Symposium*, vol. 4, pp. 447–462, San Francisco, 2011.

[42] B. Lee and J.-H. Lee, "Blockchain-based secure firmware update for embedded devices in an internet of things environment," *The Journal of Supercomputing*, vol. 73, no. 3, pp. 1152–1167, 2017.

[43] M. Li, J. Weng, A. Yang, W. Lu, Y. Zhang, L. Hou, J.-N. Liu, Y. Xiang, and R. Deng, "Crowdbc: A blockchain-based decentralized framework for crowdsourcing," *IEEE Transactions on Parallel and Distributed Systems*, 2018.