Privacy and Security in Internet of Things and Wearable Devices

Orlando Arias, *Student Member, IEEE*, Jacob Wurm, *Student Member, IEEE*, Khoa Hoang, and Yier Jin, *Member, IEEE*

Abstract—Enter the nascent era of Internet of Things (IoT) and wearable devices, where small embedded devices loaded with sensors collect information from its surroundings, process it, and relay it to remote locations for further analysis. Albeit looking harmless, these nascent technologies raise security and privacy concerns. We pose the question of the possibility and effects of compromising such devices. Concentrating on the design flow of IoT and wearable devices, we discuss some common design practices and their implications on security and privacy. Two representatives from each category, the Google Nest Thermostat and the Nike+ Fuelband, are selected as examples on how current industry practices of security as an afterthought or an add-on affect the resulting device and the potential consequences to the user's security and privacy. We then discuss design flow enhancements, through which security mechanisms can efficiently be added into a device, vastly differing from traditional practices.

Index Terms—Hardware security, user privacy, Internet of Things (IoT), wearable devices

1 Introduction

Within the past decade, the number of Internet of Things (IoT) devices introduced in the market has increased drastically. With totals approaching 15 billion, the staggering conclusion that there are roughly two connected devices per person is reached [1]. This trend is expected to continue, with an estimate of 26 billion connected devices by the year 2020, the majority of which being IoT and wearable devices [2]. Much like the embedded systems they derive from, IoT and wearable devices are armed with an array of sensors whilst also offering the means to establish a network connection, enabling the transmission of the collected information to a remote node.

The collected information can range from a simple heartbeat, to temperature and humidity data, to the location of the user himself and his living habits. As such, privacy issues arise. Also, because of the information these devices can gather and store, they become prime targets for attackers looking to obtain this data. Further, given the always on network connectivity some of these devices hold and the different usage pattern, these devices could be targeted by malware, increasing the potential for harmful usage.

While IoT manufacturers are aware of the privacy and security implications, security in IoT devices is either neglected or treated as an afterthought. This is often due to the short time to market and reduction of costs driving the device's design and development process. The few devices that do choose to add any protection usually employ software level solutions, such as firmware signing and the execution of signed binaries, methods which resemble those used in regular computing [3], [4], [5], [6], [7], [8], [9], [10], [11], [12]. These solutions, however, do not consider the different usage patterns of IoT and wearable devices compared to traditional embedded systems or personal computers, proving to be insufficient at times. Furthermore, concentrating on the software-based protection schemes often leaves the hardware unintendedly vulnerable, allowing for new attack vectors.

In order to better understand the security and privacy issues associated with current IoT device design flow and their implications, we used the Google Nest Learning Thermostat and the Nike+ Fuelband SE Fitness Tracker, hereafter referred to as the Nest Thermostat and Nike+ Fuelband, as test devices. Our selection of these units was based on the fact that both Nest Labs and Nike Inc. are among the few manufacturers who have taken steps towards securing their devices and protecting user data. Nest Labs further claims to "use best-in-class data security tools" to protect its products and user's data from unauthorized access [13]. However, as we shall demonstrate in this paper, the protection schemes used in these devices are not sufficient to secure the units themselves.

The remainder of this paper follows the ensuing organization. Section 2 introduces related work in security and privacy assurance on IoT and wearable devices. Section 3 discusses common IoT device design methodologies and possible pitfalls that may be encountered in the process. Section 4 presents our case study regarding the Nest Thermostat focusing on how to bypass software protection mechanisms using a hardware exploit. This particular attack vector and its possibilities for exploitation are then discussed in Section 5. Section 6 presents the other case study on Nike+ Fuelband. The attack vector on the wearable devices is introduced in Section 7. Further elaborating on the impact of such attacks, Section 8 explores the

• *The authors are with the Department of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL 32816. E-mail: {oarias, jacob.wurm, maximus64}@knights.ucf.edu, yier.jin@eecs.ucf.edu.*

Manuscript received 4 May 2015; revised 17 Sept. 2015; accepted 20 Oct. 2015. Date of publication 6 Nov. 2015; date of current version 11 Dec. 2015. Recommended for acceptance by S. Ray, J. Park, and S. Bhunia.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TMSCS.2015.2498605

consequences of introducing compromised IoT and wearable devices within a network. Recommended security enhancement approaches dedicated for IoT and wearable devices are discussed in Section 9. Conclusions are drawn in Section 10.

2 Related Work

Current IoT and wearable device literature often treats IoT from a network perspective or provides solutions that are inherently incompatible with the needs of a manufacturer. Few works have been published discussing the security of IoT devices themselves [14], [15]. In the ensuing sections, we summarize some of the previous work that has been presented in this area.

2.1 IoT Secure Protocols and Network Protection

An early survey about the IoT has shown that security and privacy are the main concerns that need to be addressed before IoT devices are widely adopted [16]. Proposed solutions for security rely on network protocols to ensure IoT security. Meanwhile, encrypted communication is treated as the effective solution for privacy protection. However, these proposed approaches do not consider the unique properties of IoT devices. The authors in [17] summarized all current security threats to the IoT network but these threat models are mostly derived from network security. They claim that hardware level attacks, such as differential power analysis (DPA) [18], are of high cost and therefore less harmful. Similarly, the authors in [4] treat IoT as an extremely interconnected network and list possible solutions to secure the IoT network including protocol and network security, data and privacy, identity management, trust and governance, fault tolerance, cryptography and protocols, identity and ownership, and privacy protection. All these methods try to regulate the communication between IoT devices under the assumption that all IoT devices are operating properly. The authors in [5] tried to solve IoT security through different IoT topologies: centralized architectures [6] and distributed architectures [7], [8]. Again, the network based solutions only emphasize high level structures without considering whether the available resources in IoT devices can afford these topologies.

Other research focuses on the secure communication between IoT nodes. For example, the authors in [9] focus on secure communication between IoT devices and present an Identity Authentication and Capability based Access Control (IACAC) model to protect IoT from man-in-the-middle, replay and denial of service (DoS) attacks. The authors in [10], [11] expand the definition of IoT to include four nodes in a typical IoT network: person, intelligent object, technological ecosystem, and process. The authors claim that IoT security cannot be solved at a single-layer, but should require the analysis of the interactions between these nodes. A 2D version of the systemic approach was developed, which was expanded to a 3D version highlighting new functional plans of security [12]. Following this route, communication protocols were then developed to secure the interactions between IoT nodes such as 6LoWPAN [19] and Constrained Application Protocol (CoAP) [20]. The CoAP was constructed based on Datagram Transport Layer Security (DTLS) [21] and IPsec [22]. To counter the attacks at the transport layer, protocols were enhanced to use either HTTP/TLS or CoAP/DTLS by proposing a mapping between TLS and DTLS [23] or using secure tunneling on the transport layer [24]. However, these communication layer security analyses and protection methods ignore device level vulnerabilities and often impose unrealistic constraints on device deployment.

2.2 Hardware Based Protection

Besides network level protection, researchers from the industry have also tried to develop highly secure processor/SoC architectures for IoT protection. ARM TrustZone is an industry landmark in providing a basis of trust for various applications such as secure payment, digital rights management (DRM), enterprise and web-based services. TrustZone technology provides infrastructure foundations that allow a SoC designer to choose from a range of components that can perform specific functions within the security environment [25]. Intel proposed the concept of enclaves recently [26], [27]. An enclave contains software code, data, and a stack that are protected by hardware enforced access control policies. Samsung KNOX has also been developed with protection in mind [28]. KNOX provides a safe execution environment in a KNOX-enabled device where the userland is verified and a KNOX container holds sensitive data, such as corporate contacts and e-mails in a cellphone. If the device is deemed to be compromised by altering the bootloader, an e-fuse is blown inside the SoC driving the unit, thus branding it as untrusted. However, these hardware-based secure architectures are developed with passive protection in mind, whereas they do not detect and mitigate hardware and software level attacks. Samsung KNOX is possibly an exception to this, however, it remains to be proven whether or not it is possible to bypass any checks to the e-fuse protection in the bootloader. TrustZone environments have been proven to be compromised as shown in [29], [30], [31] by exploiting bugs in the software stack. Furthermore, these solutions do not transfer well to low power embedded units. For example, at the time of writing, Samsung KNOX is only available in select Android-based cellular phones and tablets.

3 IoT Device Design Flow Practices

3.1 Reliance on Vendor Designs

Throughout our investigation of the design flow of IoT devices we have found that there are cases where the lack of familiarity with the hardware being used has led to over reliance on vendor designs. That is, products are directly based on a design or application solution a vendor has provided. Whereas for targeted applications this may be sufficient, when the only available designs are for general purpose computing devices or development boards, it may lead to the unintentional exposure of interfaces that are meant for debugging or reprogramming purposes. An attacker can easily leverage these interfaces to leak internal sensitive information or even install malicious firmware to control device operation.

3.2 Open versus Closed Source Software

At the device firmware level, it is common to find Linux-based stacks, although other devices utilize FreeRTOS [32]

or other open source projects, thus leveraging pre-existing software solutions to build upon. Other manufacturers opt for proprietary solutions, such as Wind River's vxWorks [33] or Blackberry's QNX [34]. The question of open source versus closed source software in security is a hard one to answer. With open source software, an attacker just needs to find a potential vector to target the device by looking for errors in the source code. However, a manufacturer does not have to rely on the system's vendor in order to patch the bug, thus enabling a faster response time. With a closed source system, however, an attacker is faced with a problem in reverse engineering interfaces looking for potential errors in the software stack. Manufacturers, however, need to rely on vendors once vulnerabilities are found. The stack chosen should then be selected based on design requirements, availability of support, documentation and amount of security offered.

3.3 Weak or Bad Cryptographic Implementations

If a device is designed to be remotely updated, it must be able to verify the downloaded image for both integrity and authenticity. This usually involves a cryptographic algorithm, sometimes many. Cryptographically securing a product is a complicated task, as proven by the countless vulnerabilities found in software, not only because of the mathematics involved, but because of implementation errors [35], [36], [37], [38], [39], [40]. Two of these vulnerabilities are of critical importance to our research as it shows how weakly implemented cryptographic systems can be bypassed, providing a way to remotely attack the device. These exploits describe how an attacker can remotely compromise a Belkin WeMo Home Automation device by exploiting the faulty usage of SSL, allowing remote firmware installation by spoofing a distribution server, or by spoofing SSL servers via arbitrary certificates.

3.4 Debug Interfaces on Production Runs

It is often cheaper to write images to flash chips when assembling the device, rather than purchasing preprogrammed parts. Furthermore, the device must be functionally tested before it leaves production. This implies that the circuit board must expose programming interfaces and test points for the different components present within. Although at times unlabeled, these often unpopulated interfaces are not removed after testing. An attacker can utilize them to inject his own code on the unit or alter their functional behavior. The software component may also fall prey to this issue, as compilers can generate binaries that include debugging symbols, expressing the constructs that generated a certain block of machine code. Leaving these debugging symbols in production runs aids an attacker in reconstructing the original sources, allowing for easier vulnerability detection.

3.5 Supply Chain Threats

Hardware Trojans also pose a serious threat to IoT security. These malicious modifications to integrated circuits can leak key data to an attacker, cause a device to operate outside specified parameters, or otherwise render the device inoperable. Hardware Trojans further pose the threat of not being detected by normal testing methodologies, requiring expensive specialized tests to detect them. For example, a malicious adversary could insert a hardware Trojan in a cryptographic IP core utilized in a system-on-chip (SoC) used in an IoT device [41]. When triggered, this Trojan weakens the entropy of the random number generator used to generate keys. If these keys are used to encrypt sensitive data that is being transmitted by the device, the amount of computational effort required by the attacker to decrypt the data is severely reduced.

4 Case Study 1: Nest Thermostat

As part of our research, we present the Nest Thermostat as a case study. We disassembled the device and explored its functionality with the objective to find any vulnerabilities that were left in the hardware and software stack.

4.1 High Level Overview

The Nest Thermostat is a smart device designed to control a standard heating, ventilation and air conditioning (HVAC) unit based on heuristics and learned behavior. The thermostat is also equipped with a motion sensor capable of detecting whether users are at the installed location and control the HVAC unit accordingly. Coupled with a WiFi module, the unit is able to connect to the user's home or office network and interface with the Nest Cloud, thereby allowing for remote control of the unit. It also exhibits a ZigBee module for communication with other Nest devices, but has remained dormant as of firmware versions up to the current 4.2.x series.

The Nest Thermostat runs a Linux kernel, coupled with some GNU userland tools, Busybox, other miscellaneous utilities in order to run a proprietary stack designed and written by Nest Labs. To remain GPL compliant, the modified source code used within the device has been published and is available for downloading from Nest Lab's Open Source Compliance page [42], with the notable exception of the C library. Build scripts to generate binaries from these sources are provided, whereas a toolchain was unavailable to users until shortly after preliminary results of our research were presented.

Energy savings are attempted by gathering usage statistics and environmental factors to systematically build a user profile. As these metrics are coupled with user input, they provide a comfortable environment. The profile is also uploaded to Nest Cloud, a service where users can remotely interact with their device. The manufacturer proceeds to gather and study this information with the hopes of aiding energy providers with the means to achieve optimal energy generation.

4.2 Device Security

The Nest Thermostat contains two wireless communication channels, a WiFi interface and a ZigBee interface. At the time of writing, only the WiFi interface is active and used. The Thermostat is capable of connecting to wireless networks encrypted using WPA2–Personal but it is incapable of connecting to WPA2–Enterprise encrypted networks. Other legacy connection standards are also supported. Any log-related communication started by the unit is encrypted using TLS 1.2 from the beginning, making it hard to intercept any data that is being transmitted by these means.

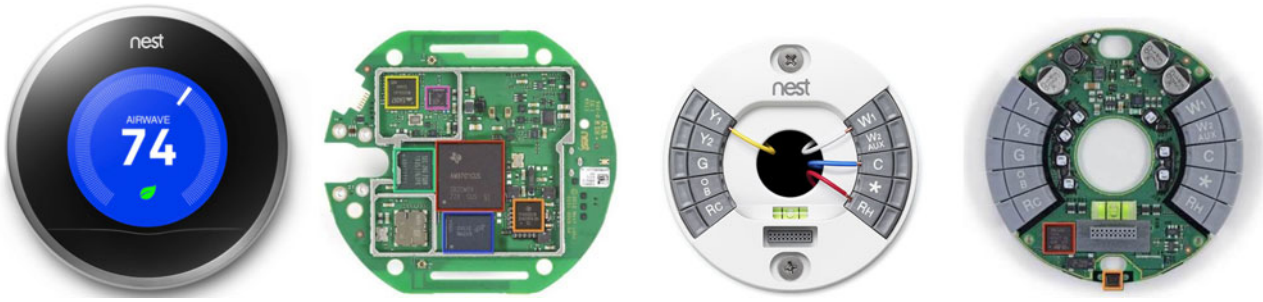


Fig. 1. Front (left) and backplate (right) of a Nest Thermostat (credit: Nest, iFixit).

This includes access to the Nest Cloud, which authenticates credentials using OAuth tokens. OAuth tokens have the advantage that they can easily be revoked by the issuer (in this case, Nest Cloud) and can be used to limit access to certain account features. Updates and non-critical data such as weather is obtained over a plain-text communication channel. This data can be potentially intercepted and modified. However, update images are cryptographically signed using PKCS #7 certificates, thus modifying an update image results in invalidating the cryptographic signature. The Nest Thermostat's internal software stack rejects any update image which does not contain a valid signature.

4.3 Device Descriptive Overview

The thermostat is divided into two main components, a backplate which interfaces with the HVAC unit and a front plate which presents the main user interface.

The backplate is managed by a ST Microelectronics ARM Cortex-M3 based microcontroller. An SHT20 temperature sensor communicates with the microcontroller using the I^2C bus protocol. A rectifier bridge and switching supply is used in order to retrieve power from the HVAC unit. A few driver circuits are also present, as to manipulate the control signals utilized by most HVAC systems. The backplate contains a 2 by 20 connector which provides access to some of the microcontroller peripherals, such as UART, power rails and other control signals.

The largest part count is found in the front plate of the thermostat, which is driven by a Texas Instruments Sitara AM3703 system-on-chip [43], interfacing directly with a Micron ECC NAND flash memory module, a Samsung SDRAM memory module and a LCD screen. The front plate also holds two wireless connectivity modules (an Ember EM3567 for ZigBee and a TI WL1270B coupled with a Skyworks SKY2463 for WiFi), a button, a long range and a short range motion sensor, an optical navigation module (ADBM-A350) and other miscellaneous components. Power distribution within the front plate is managed by a Texas Instruments TPS65921B power management module, which also provides high speed USB capabilities. A customized GNU/Linux stack provides the backbone of the software interface, with our research units running kernel version 2.6.37. Figs. 1 and 2 show the device internals and device map, respectively.

4.4 The AM3703—A Close Look

The TI AM3703 SoC is composed of a 32 Channel DMA controller, a dual-output three-layer display processor, High Speed USB controller with USB OTG capabilities, an emulation module for debugging, a General Purpose Memory Controller (GPMC) to handle NAND/NOR flash, an SDRAM memory scheduler and controller, an 112 KiB on-chip ROM which contains boot code, a 64 KiB on-chip SRAM all connected by a Level 3 (L3) interconnect which runs at 200 MHz. The ARM core within the MPU subsystem uses a 256 KiB cache to reach the L3 interconnect. Furthermore, a Level 4 interconnect adds the peripheral module to the memory map. This peripheral module handles the GPIO, UARTs, high speed multimaster I^2C bus, memory card controller, memory stick pro controller, watchdog timer, general purpose timers and other miscellaneous subsystems [43].

The ARM subchip integrates an ARM Cortex-A8 core, with Version 7 of the instruction set architecture, providing standard ARM instructions and Thumb-2 mode, the JazelleX Java accelerator and media extensions. It also integrates an ARM NEON core SIMD coprocessor. The subchip connects to 32 KiB/32 KiB instruction/data caches which proceeds to interface with a 256 KiB eight-way associative cache supporting parity and ECC. The core also provides integrated trace and debug features [43]. A simplified memory map of the AM3703 is shown in Fig. 3.

4.5 Boot Process and Device Initialization

Upon normal power on conditions, the Sitara AM3703 starts to execute the code in its internal ROM. This code initializes the most basic peripherals, including the General Purpose Memory Controller. It then looks for the first stage bootloader, `x-loader`, and places it into SRAM. Once this

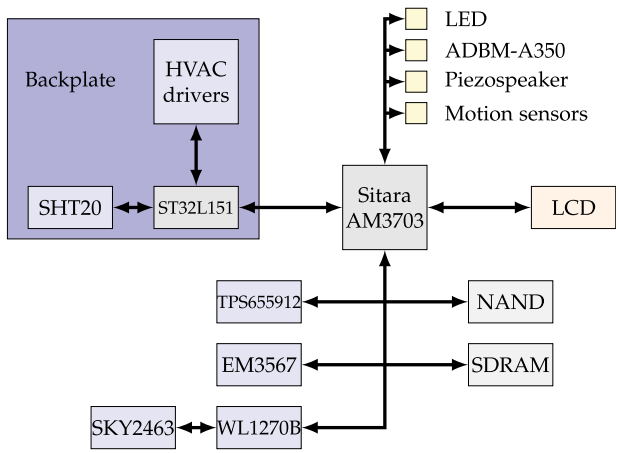


Fig. 2. Device map of the Nest Thermostat.

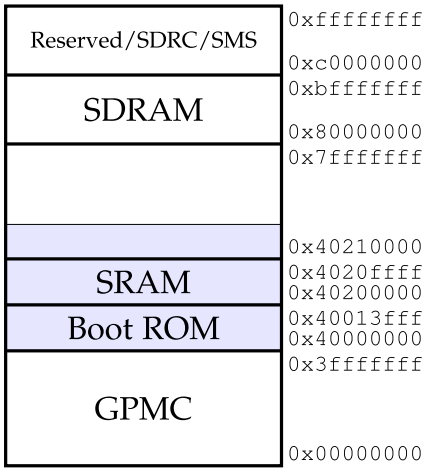


Fig. 3. Simplified memory map (shaded areas are internal to the AM3703).

operation finishes, the ROM code jumps into `x-loader`, which proceeds to initialize other peripherals and SDRAM. Afterwards, it copies the second stage bootloader, `u-boot`, into SDRAM and proceeds to execute it. At this point, `u-boot` initializes the remaining subsystems and executes the `uImage` in NAND flash with the configured environment. The system finishes booting from NAND flash as initialization scripts are executed and services are run, culminating with the loading of the Nest Thermostat proprietary software stack. Fig. 4 shows the normal boot sequence of the device.

Power connections, clock and reset signals must be properly initialized before the AM3703 boots. The device boot configuration is given by six external pins, `sys_boot[5:0]`. After power-on reset, the value on these pins are latched into the `CONTROL.CONTROL_STATUS` register. Table 1 describes the boot selection process for a selected set of configurations.

After performing basic initialization tasks, the on-chip ROM may jump into a connected execute in place (XIP) memory, if the `sys_boot` pins are configured as such. This boot mode is executed as a blind jump to the external addressable memory as soon as it is available. Otherwise, the ROM constructs a boot device list to be searched for boot images and stores it in the first location of available scratchpad memory. The construction of this list depends on whether or not the device is booting from a power-on reset state. If the device is booting from a power-on reset, the boot configuration is read directly from the `sys_boot` pins and latched into the `CONTROL.CONTROL_STATUS` register. Otherwise, the ROM will look in the scratchpad area of SRAM for a valid boot configuration. If it finds one, it will utilize it, otherwise it will build one from *permanent devices* as configured in the `sys_boot` pins. The flowchart in Fig. 5 provides a graphical view of this process.

TABLE 1
Selected Boot Configurations

`sys_boot[5:0]`	First	Second	Third	Fourth	Fifth
001101	XIP	USB	UART3	MMC1	
001110	XIPwait	DOC	USB	UART3	MMC1
001111	NAND	USB	UART3	MMC1	
101101	USB	UART3	MMC1	XIP	
101110	USB	UART3	MMC1	XIPwait	DOC
101111	USB	UART3	MMC1	NAND	

5 Attack Vector on Nest Thermostat

Examination of the circuit board for the Nest Thermostat shows that the `sys_boot[5]` pin is not only exposed in a pad, but also in an unpopulated header. As shown before, if this pin is pulled high, the processor is made to boot from a peripheral interface, namely USB or UART3. This behavior can be exploited to insert our own code into the device. Furthermore, by pressing the button in the thermostat for about 10 seconds, it is possible to trigger a hard reset of the device causing the `sys_boot[5]` pin to be driven high, ensuing the same behavior. Since ROM does not run any kind of verification on the code being injected, we are able to run it without restriction. Only the timing windows for device detection and programming must be met. The first payload must be `x-loader`, which is copied into SRAM. Subsequent payloads are copied into SDRAM.

5.1 Initial Attack

Our initial attack consisted of sending `x-loader` into the unit by means of USB, coupled with a custom `u-boot` crafted with an argument list to be passed to the on-board kernel. Our `u-boot` image was sent along a custom `ramdisk` which contained the final payload. As the on-board kernel booted, our `ramdisk` was utilized as the `init` filesystem. This gave us a very rustic shell upon which to modify the device's behavior. By mounting the device's filesystem, we enabled remote access to the unit using the already onboard `netcat` binary. The device's `init` script was also modified in order to activate this shell upon any subsequent boot.

Reconstruction of the userland by means of the shell allowed us to start initial forensic analysis of the device. This ensued in obtaining toolchain information, Application Binary Interface (ABI) information and other missing areas

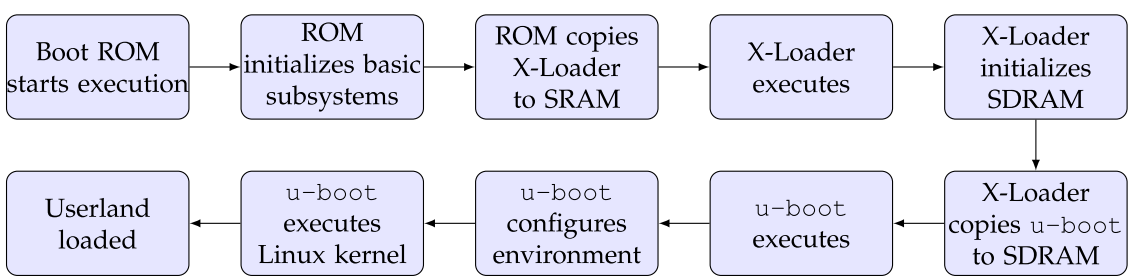


Fig. 4. Standard Nest Thermostat boot process.

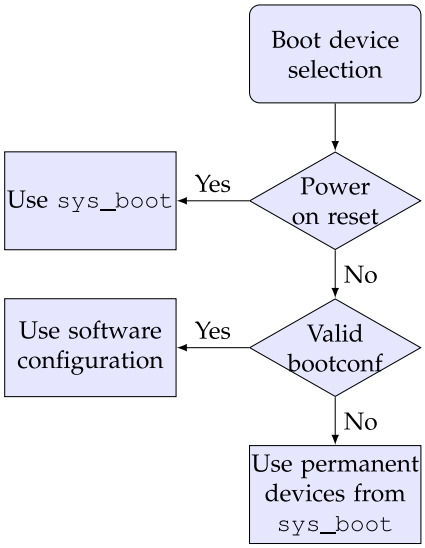


Fig. 5. Boot device setup.

of the userland. With the information at hand, a full toolchain was developed which enabled us to build arbitrary payloads targeting the Nest Thermostat.

5.2 Refining Backdoors

With the new tools, the `dropbear` secure shell and SFTP server was cross compiled, providing us with a better way to access the unit, after making the respective changes to the system's `/etc/passwd`, `/etc/shadow` and `/etc/groups` files, enabling a user account.

With secure shell having been enabled, accessing the device within the local network became easier and more reliable. Further forensic analysis of the unit was performed this way, discovering the storage of all logged data and the possibility of its retrieval by unauthorized sources.

5.3 Dialing from Ilion

Under regular operating conditions, the Nest Thermostat is behind a Network Address Translation (NAT) firewall, meaning that accessing it from a remote location by means of standard secure shell requires a user to enable port forwarding in their firewall. As such, we developed a proof of concept Trojan horse, codenamed *Odysseus* which is designed to dial into a remote server, *Achaea*, and await commands. The Trojan was injected into a thermostat unit and deployed into a proof of concept smart house we named *Ilion*. Within its network, *Ilion* contained laptop and desktop computers, smartphones, tablets and a few other devices, as to emulate a real life setting.

Since the Nest Thermostat was now a part of *Ilion*, we were able to utilize *Odysseus* to extract the network credentials. *Odysseus* was also used to scan the network for other devices, sending this information to *Achaea*, where the remote attacker can collect it. Fig. 6 shows traffic from a session of *Odysseus* relaying collected information. Our Trojan also enabled us to deploy a rogue DHCP server, allowing us to shape the outgoing traffic by redirecting DNS requests to our servers, thus enabling us to launch a wider variety of attacks against other connected devices.

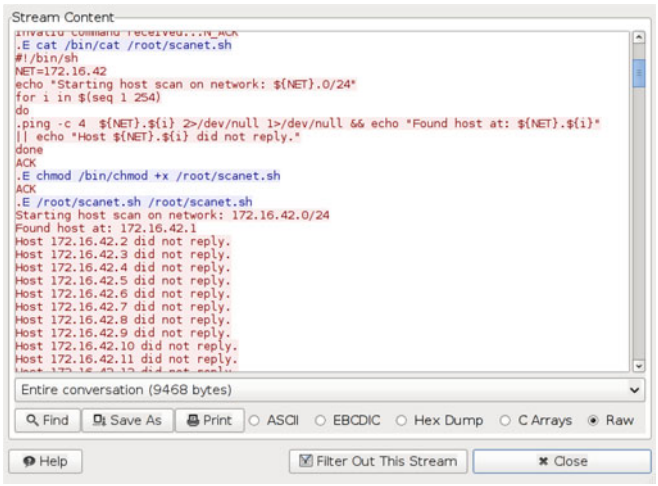


Fig. 6. Sample traffic between Odysseus and Achaea.

5.4 Remote Updates and the Linux Kernel

The Nest Thermostat receives signed updates from the Nest website over standard, plain-text HTTP. As such, the connection can be eavesdropped and images can be downloaded to an alternate location and analyzed for contents. The update images are not encrypted, but they contain a manifest file with signing keys. These signatures are verified against a public certificate found on the device. It is possible then to alter the certificate verification within a compromised unit to accept updates from a different source.

With our current toolset, we were able to obtain the kernel configuration file generated within the `/proc` filesystem. Further study of the device can be accomplished by means of intercepting system calls and kernel level debugging. As such, using this configuration file as a base, a custom kernel was built with debugging features added. Since `u-boot` must provide the layout of NAND flash to the kernel, analysis of its sources yielded a possible place of storage for our customized kernel.

Our custom kernel has been patched to allow polling in the OMAP serial driver, allowing us to use the kernel level debugger `kgdb` through one of the exposed serial ports. We have permanently written this kernel to the `boot1` section of NAND flash and use a custom `u-boot` to select whether to use the stock kernel or ours upon powering on the unit. With a custom kernel in place, as mentioned above, we can intercept system calls, potentially disabling reads and writes to specific sections of NAND flash. This enables us to block certain files from being deleted, even if we allow official Nest updates to run, giving persistence to any software backdoors injected within the device.

6 Case Study 2: Nike+ Fuelband

Architecture wise, wearable and medical devices resemble IoT devices, however, they tend to have much less computational power and limited communication interfaces. Nevertheless, these units perform as much if not more data collection than IoT devices do. Although closely related to IoT devices, security vulnerabilities on wearable devices can lead to safety concerns for users. A pacemaker with wireless capabilities was proven to be vulnerable and could be used to affect the health of the patient [44]. Information leaks from

Fig. 7. Nike+ Fuelband SE Fitness Tracker (credit: Nike).

fitness devices owned by corporate executives could be used against them, causing the corporation's value to deteriorate on the market, severely affecting its performance.

Much like our work with the Nest Thermostat, we performed a similar analysis on medical and wearable devices, looking for possible hardware vulnerabilities which may be utilized against an unsuspecting user. In the following sections, we introduce as a secondary case study our work with the Nike+ Fuelband, a wearable device with fitness monitoring capabilities.

6.1 High Level Overview

The Nike+ Fuelband is a low-power Bluetooth 4.0-enabled fitness wristband designed to measure daily physical activity, such as the amount of steps taken, sleep patterns and estimate the amount of calories burned (see Fig. 7). This is done by means of reading data from the on-board three-axis accelerometer, which is subsequently stored within the unit. By means of software provided by the manufacturer, the unit can communicate with a Windows or OS X based computer, as well as Android and iOS devices. The collected data can then be analyzed, tracked and shared with the Nike+ online community. Periodic synchronization with the device can be achieved with the mobile applications and real-time feedback is performed with the on-board LED matrix display. The device is powered by two Lithium-polymer batteries, advertised to provide up to four days of continuous usage.

6.2 Device Security

The Nike+ Fuelband contains a Bluetooth interface which it uses to communicate with a smartphone. Some settings of the Fuelband can be configured through these means and information from the band can be sent back to the smartphone using this channel. Firmware updates, however, are performed by means of the Nike+ application on a Windows or OS X based personal computer. Most of the communications from the smartband are done through the smartphone or personal computer application. Upon boot, the firmware is checked against a checksum before it is run ensuring a valid image.

6.3 Device Descriptive Overview

The main processing unit in this device is the ST Microelectronics STM32L151QCH6 microcontroller. Built upon an ARM Cortex-M3 core, this microcontroller is described in greater detail in Section 6.4. An LIS3DH three-axis MEMS accelerometer from the same manufacturer interfaces with the STM32 by means of a Silego SLG46300 programmable mixed signal array. The 120-LED matrix is driven by an AMS AS1130 driver, which simplifies some LED matrix related operations. Power management is provided by the ST Microelectronics RS12, which also facilitates communications over USB 2.0. Bluetooth communication is achieved by means of a Cambridge Silicon Radio CSR1010 Bluetooth Low Energy module. Fig. 8 shows the device map of the unit.

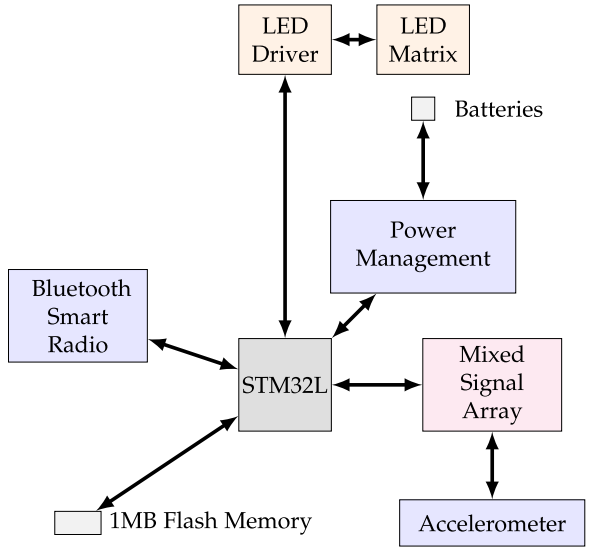


Fig. 8. Device map of the fuelband.

6.4 The STM32L151QCH6—A Closer Look

The ST Microelectronics STM32L151QCH6 system on a chip, hereafter referred to as STM32, is an ultra-low-power platform offering a 12 channel DMA controller, 23 capacitive sensing channels and a CRC calculation unit. The SoC further includes a 96 bit unique ID, a preprogrammed bootloader supporting both USB and USART programming, 116 fast input/output pins which are mappable to a 16 interrupt vector table. Storage wise, the STM32 in question offers 256 KiB of flash storage with ECC support, 32 KiB of SRAM, 8 KiB of ECC supporting EEPROM and a 128 B backup register. Included peripherals range from an LCD driver, to communication interfaces supporting USB 2.0, USART, SPI and I^2C [45].

The included ARM Cortex-M3 core supports both the Thumb and Thumb-2 instruction set architectures. Advanced low-power optimizations are achieved by means of multiple power and clock domains, architecture defined sleep modes and support for advanced low-power technologies such as State Retention Power Gating. A JTAG mechanism is provided by means of serial wire debug, which provides real-time access to system memory without halting the processor.

A simplified memory map of the STM32 is illustrated in Fig. 9. The highlighted block of addresses in the figure are multiplexed between Flash or System Memory, depending on the status of the external `BOOT0` pin (see Section 6.5).

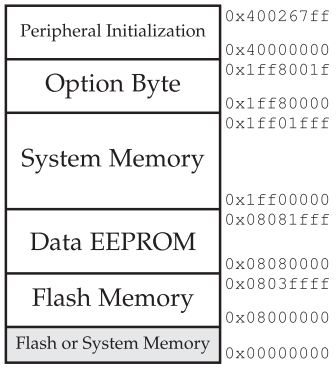


Fig. 9. Simplified memory map of the STM32L151QCH6.

6.5 Boot Process and Device Initialization

Upon device power on, the STM32 executes the code stored in its internal ROM, initializing the device's basic peripherals. Execution then continues from internal flash memory, which proceeds to finish device setup into a working model. Specific to the Nike+ Fuelband, this entails activation of the Bluetooth radio, mixed signal array and LED driver, along with the calibration of the accelerometer. At this point, the device is ready for regular usage.

The STM32, however, implements a secondary boot mode, which is triggered by holding the `BOOT0` pin to a logic 1 as the device starts. If started this way, the device initializes a basic set of peripherals and configures the USB subsystem. Then, if a USB cable is detected whilst being driven by the proper clock signal, the internal PLL reconfigures the system clock to 32 MHz and the USB subsystem clock to 48 MHz. The system proceeds to execute the DFU bootloader with USB interrupts enabled, as to allow for communication. Using this mechanism, the STM32 can be sent commands which allow for read and write operations to memory, changing memory protection modes and status retrieval.

7 Attack Vector on the Nike+ Fuelband

Although the STM32 documentation states that the microprocessor contains the necessary capabilities to lock external reads and writes against the internal flash, thus isolating the device's firmware from the external world, this protection was not employed on the Nike+ Fuelband. As such, the contents of flash can be freely modified by an attacker with access to the device.

The Nike+ Fuelband contains a standard USB connector which is used for both device charging and synchronization. This connector can also be used to write new firmware onto the device, however, the necessary access to the `BOOT0` pin is not externally provided. As such, the device must be opened in order to trigger the alternate boot sequence. Further complicating the issue is the fact that the microcontroller is packaged as a Ball Grid Array (BGA) and thus no direct access to the `BOOT0` pin can be obtained. Traces on the circuit board must then be followed in order to encounter a test point indirectly exposing the pin in question.

After following this process, we were able to indirectly locate the `BOOT0` pin, which was subsequently driven to a logic 1 state by means of a 100 Ω resistor connected to V_{DD}. This allowed us to enter the alternate boot mechanism and exploit the lack of read and write protection on the device. By means of standard ST Microelectronics development tools, communication over USB with the STM32 was achieved and the device's firmware was obtained.

With the device's firmware in our hand, we set on to modify it. The simplest change is one of string replacement, that is, find a string in the program that gets displayed at some point and change it to something else. With the change made, the modified firmware was written to the device, only to find normal functionality had ceased to exist. Further testing demonstrated that this was caused by a failure to compute the proper CRC for the image. Since the image was modified, the check failed.

Closer examination of the disassembled firmware image demonstrated that it utilized the CRC engine within the STM32 microcontroller in order to verify itself as genuine by checking the result of the CRC computation against a stored value. This value was found within the image itself, and thus easily modifiable. With the proper checksum added, the modified firmware was sent to the device and proven to work.

8 Discussions

8.1 Security Impact to Network

A compromised IoT device can be utilized to further attack other units in an unsuspecting victim's network. Effects could range from simple backdoor injection to leaking user information and credentials to even causing physical harm to the user. As shown with the case of the Nest Thermostat, it can be used as a beachhead to other nodes within the network, allowing for discovery and attack of those nodes.

Furthermore, rogue services may be installed on the device, aiming to disrupt regular network operations. For instance, a rogue DHCP server may be utilized to inject DNS requests to a poisoned server which would return false information, allowing for traffic shaping. Address Resolution Protocol (ARP) based attacks are also possible, with the compromised device masquerading as the router, allowing for the capture and redirection of a target computer's network traffic.

Security issues with backdoored IoT devices are exacerbated by the fact that local network credentials need to be stored within the unit, thus becoming accessible to an attacker. Leveraging the extraction of network credentials allows for the introduction of extraneous devices into the local network, granting for new methods of exploitation against other nodes. In the case of the Nest Thermostat, the network credentials are stored in regular text files, and even if these were encrypted, the algorithms necessary to obtain the clear text would necessarily be present on the device, granting the attacker the means to collect them.

8.2 Safety Concerns

Safety concerns arise when compromised IoT and wearable devices see on-field deployment. Due to the services these units provide, from communications to medical applications, a compromised device could then be used to cause physical harm to its user [44]. The Nest Thermostat could be employed to overstress the HVAC unit it is connected to, causing it to malfunction. Furthermore, all the information stored within the device can be utilized by the attacker to

build a profile of the victim, aiding in the determination of a daily routine, the usage of which can result in facilitating the burglarizing of the victim's property.

8.3 Privacy Concerns

Almost all IoT and wearable devices, upon setup, will start collecting user information. For example, the Nest Thermostat will collect information such as the location of the thermostat, whether it is being used in a home or business, the postal code of the area and device information from the HVAC system to determine its capabilities. The on-board sensors on the thermostat will also collect temperature data, humidity and ambient light data, and by means of the onboard passive infrared sensor, whether somebody is moving in the room. Any direct temperature adjustments to the device are also recorded and utilized in algorithms to learn and compute comfort levels under different situations. Whenever the HVAC unit is activated, the thermostat will record the time and duration for which this happened. Using this information, the thermostat builds a profile for the users in order to help them feel comfortable whilst also providing energy savings. The Nike+ Fuelband will store the user's heartbeat and sleeping patterns, which can then be learned by the attacker. The information could potentially be used against the user, or against any entity the user is part of.

Although there are laws and standards defining data collection policies, some of these have proven to be ineffective and are often antiquated, as demonstrated by information leaks from companies [46], [47], [48]. User information collected by the Nest Thermostat is stored within the unit and uploaded to the Nest Cloud. Local log files are sent to Nest as well and removed from the unit as to save space. System and software logs contain information such as the user's Zip code, device settings, HVAC settings and wiring configuration. Forensic analysis of the unit yields that the Nest Thermostat has code to prompt the user for information about their place of residence or office. Reports indicate that Nest plans to share this information with energy providers in order to aid with efficient power generation [49]. As for the Nike+ Fuelband, the information collected and stored by the unit is then sent to a personal computer or mobile device, from where it can be publically shared with other users. Even if the information is not shared, an unauthorized third party still has access to the data from a compromised device and can use it for their own purposes. Although IoT manufacturers have gone through considerable efforts to ensure the secure transmission of this data, it is all for naught if it can be leaked at the source.

9 Device Security Enhancement

9.1 Security Solutions Common to IoT and Wearable Devices

Verifying the firmware at update time is a step towards securing IoT devices, however, this is often done by the on-board software. As with the Nest Thermostat and the Nike+ Fuelband, the on-board software is trusted to be authentic. The implementation of this check, however, must be sound. For example, schemes that utilize random numbers must ensure the usage of a cryptographically secure random number generator, any used cryptographic certificates must be validated by a trusted Certificate Authority [39]. A weakly implemented cryptographic algorithm is no better than a lack of a cryptographic algorithm.

However, as we have demonstrated with our case studies, it is insufficient to authenticate an update image. The software stack must also be authenticated before it can reliably determine if an update is valid or not. With the devices compromised, we are free to bypass any checks on the update image, thus rendering the protection mechanism ineffective. A proper chain of trust in the hardware infrastructure of the device can aid the process of determining an authentic software stack [50].

The attack in both the Nest Thermostat and the Nike+ Fuelband could have been avoided had a proper chain of trust been implemented. Inherently, this needs the type of hardware support which is not available in either the Sitara AM 3703 used in the Nest Thermostat or the STM32 microcontroller used in the Nike+ Fuelband.

The exposure of debug interfaces in these devices further presents a risk. These are often left as residues from development prototypes or as testpoints used during manufacturing. These debug interfaces can also serve as the means to service IoT or wearable devices on the field, as to ease repairs. As such, we can see why they may be needed. However, these interfaces must be protected against attackers. For example, FRAM devices in the MSP430 lines provide means to both secure JTAG access and to protect certain memory segments from access using a built in IP Encapsulation Module [51]. Other microcontrollers and microprocessors offer the same kind of functionality, implementing means to restrict access to its debug units. As such, manufacturers are able to still expose these interfaces for testing purposes and lock them before they are deployed. Ideally, however, any debug interfaces should be removed from production runs or have proper protections.

9.2 Specific Solutions for IoT Devices

Often, IoT devices provide a full operating system in which binaries are loaded into an userland. This simplifies the interface to the hardware and provides high level Application Programming Interfaces (APIs). The Nest Thermostat, for example, employs an embedded Linux stack which is used to launch the proprietary Nest application which relays commands to the backplate of the unit and controls the communications channels. As we demonstrated in our case study, binaries can be injected into the filesystem of the unit and executed in devices that utilize this model. As such, extra protection must be added to devices that load binaries into a userland. A possible approach is to only load and execute cryptographically signed binaries. This requires the kernel to have a custom loader that verifies these binaries as they are prepared for execution. If the signature verification fails, then the binary is not run and the device is set into a failsafe mode, notifying the user of possible tampering.

9.3 Specific Solutions for Wearable Devices

In devices whose architecture is self contained, that is, microcontroller based systems, it becomes necessary to secure all update channels. External reprogrammability of

the microcontroller and any debug interfaces it may feature must be disabled. The microcontroller must also be programmed before being placed in the circuit board, as to avoid adding unnecessary interfaces which could expose functionality.

9.4 Overhead of Security Solutions

There is usually a certain degree of overhead associated with any protection mechanism. Cryptography necessarily adds computational overhead to any protection scheme that utilizes it. It may be reasonable to expect then that any device which utilizes encryption or any other cryptographic function to require binaries with functions to include the necessary checks and have higher memory and CPU requirements in order to perform better. However, current industry solutions include parts which are capable of accelerating these processes, much like the microcontroller utilized in the Nike+ Fuelband which can accelerate CRC32 computations [45]. This reduces the software overhead needed to perform these checks, but slightly increases the area and power consumption of these parts. It should be noted, however that for most parts, power can be gated to the SoC subsystems that are not being utilized, thus reducing power consumption in the device.

10 Conclusions and Future Work

As our case studies demonstrated, a non-secure hardware platform will inevitably lead to a non-secure software stack. A vulnerability in the design of the unit can result in its compromise. Furthermore, without being able to authenticate the running software, it can not be trusted to make decisions about its own validity. Due to the short time to market engineers are given to finish a product, we believe that most of the current IoT and wearable devices suffer from similar issues. Software protection becomes ineffective if the hardware is vulnerable to attack. This raises safety and privacy issues with users, is their information safe?

Moving forward, we will continue to probe other IoT devices for security, with the goal of finding vulnerabilities in their hardware. Ultimately, this will lead us to a better understanding of design issues and how to correct them. We will attempt to build prototypes of smart devices that utilize our proposed chain of trust to test for their viability and ability to prevent malicious attacks.

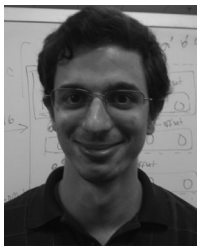
Acknowledgments

Mr. Orlando Arias and Mr. Khoa Hoang are partly supported by the REU Supplement of the US National Science Foundation (NSF) award (CNS-1319105).

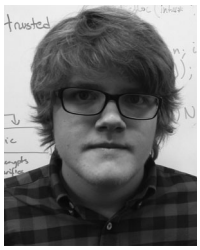
References

[1] D. Evans, "The internet of things - how the next evolution of the internet is changing everything," *White Paper. Cisco Internet Business Solutions Group (IBSG)*, 2011.
[2] P. Middleton, P. Kjeldsen, and J. Tully, "Forecast: The internet of things, worldwide, 2013," *Gartner*, 2013.
[3] D. Welch and S. Lathrop, "Wireless security threat taxonomy," in *Proc. IEEE Syst., Man Cybern. Soc. Inf. Assurance Workshop*, 2003, pp. 76–83.
[4] R. Roman, P. Najera, and J. Lopez, "Securing the internet of things," *Computer*, vol. 44, no. 9, pp. 51–58, Sep. 2011.
[5] R. Roman, J. Zhou, and J. Lopez, "On the features and challenges of security and privacy in distributed internet of things," *Comput. Netw.*, vol. 57, no. 10, pp. 2266–2279, 2013.
[6] A. Williams. (2011). How the internet of things helps us understand radiation levels [Online]. http://readwrite.com/2011/04/01/ow-the-internet-of-things-help.
[7] D. Viehland and F. Zhao, "The future of personal area networks in a ubiquitous computing world," *Int. J. Adv. Pervasive Ubiquitous Comput.*, vol. 2, no. 2, pp. 30–44, 2010.
[8] H. Schaffers, N. Komninos, M. Pallot, B. Trousse, M. Nilsson, and A. Oliveira, "Smart cities and the future internet: Towards cooperation frameworks for open innovation," in *The Future Internet*, Berlin, Germany: Springer, 2011, pp. 431–446.
[9] P. N. Mahalle, B. Anggorojati, N. R. Prasad, and R. Prasad, "Identify authentication and capability based access control (IACAC) for the internet of things," *J. Cyber Security Mobility*, vol. 1, pp. 309–348, 2013.
[10] Y. Challal, "Internet of things security: Towards a cognitive and systemic approach," Ph.D. dissertation, Compiegne University of Technology, 2012.
[11] A. Riahi, Y. Challal, E. Natalizio, Z. Chtourou, and A. Bouabdallah, "A systemic approach for IoT security," in *Proc. IEEE Int. Conf. Distrib. Comput. Sensor Syst.*, 2013, pp. 351–355.
[12] A. Riahi, E. Natalizio, Y. Challal, N. Mitton, and A. Iera, "A systemic and cognitive approach for IoT security," in *Proc. Int. Conf. Comput., Netw. Commun.*, 2014, pp. 183–188.
[13] (2015). Nest Labs. Privacy statement [Online]. https://nest.com/legal/privacy-statement/
[14] J. H. Ziegeldorf, O. G. Morchon, and K. Wehrle, "Privacy in the internet of things: Threats and challenges," *Security Commun. Netw.*, vol. 7, no. 12, pp. 2728–2742, 2014.
[15] A. D. Thierer, "The internet of things and wearable technology: Addressing privacy and security concerns without derailing innovation," *Rich. J. Law Technol.*, vol. 21, pp. 6–15, 2015.
[16] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Comput. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.
[17] S. Babar, P. Mahalle, A. Stango, N. Prasad, and R. Prasad, "Proposed security model and threat taxonomy for the internet of things (IoT)," in *Proc. Recent Trends Netw. Security Appl.: 3rd Int. Conf.*, 2010, pp. 420–429.
[18] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Proc. Adv. Cryptol.*, 1999, pp. 789–789.
[19] G. Mulligan, "The 6LowPAN architecture," in *Proc. 4th Workshop Embedded Netw. Sensors*, 2007, pp. 78–82.
[20] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained application protocol (CoAP), draft-ietf-core-coap-13," in *The Internet Engineering Task Force (IETF)*, 2012.
[21] E. Rescorla and N. Modadugu, "Datagram transport layer security," *RFC 4347*, 2006.
[22] S. Kent and K. Seo, "Security architecture for the internet protocol," *RFC 4301*, 2005.
[23] M. Brachmann, S. L. Keoh, O. Morchon, and S. Kumar, "End-to-end transport security in the ip-based internet of things," in *Proc. 21st Int. Conf. Comput. Commun. Netw.*, 2012, pp. 1–5.
[24] R. Seggelmann, "Sctp: Strategies to secure end-to-end communication," Ph.D. dissertation, Univ. Duisburg-Essen, Essen, Germany, 2012.
[25] ARM, "Building a secure system using trustzone technology," ARM Limited, Cambridge, England, 2009.
[26] F. McKeen, I. Alexandrovich, A. Berenzon, C. Rozas, H. Shafi, V. Shanbhogue, and U. Savagaonkar, "Innovative instruction and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architectural Support Security Privacy*, 2013.
[27] I. Anati, S. Gueron, S. P. Johnson, and V. R. Scarlata, "Innovative technology for cpu based attestation and sealing," in *Proc. 2nd Int. Workshop Hardware Architectural Support Security Privacy*, 2013.
[28] Samsung. (2015). Samsung KNOX: Mobile Enterprise Security [Online]. Available: https://www.samsungknox.com/en
[29] N. Keltner and C. Holmes, "Here be dragons: A bedtime tale for sleepless nights," in *RedCon*, 2014.
[30] D. Rosenberg, "Reflections on trusting trustzone," in *BlackHat USA*, 2014.
[31] T. Wei and Y. Zhang, "To swipe or not to swipe: A challenge for your fingers," in *Proc. RSA Conf.*, 2015.
[32] "Freertos reference manual: API functions and configuration options," Real Time Eng. Limited, Tech. Rep. WC2H 9JQ, London, England, 2009.

[33] A. Barbalace, A. Luchetta, G. Manduchi, M. Moro, A. Soppelsa, and C. Taliercio, "Performance comparison of VxWorks, Linux, RTAI and Xenomai in a hard real-time application," in *Proc. 15th IEEE-NPSS Real-Time Conf.*, 2007, pp. 1–5.
[34] (1982-2014). QNX operating systems [Online]. Available: http://www.qnx.com/products/neutrino-rtos/index.html
[35] CVE-2014-0160. Common Vulnerabilities and Exposures [Online]. Available: https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-0160
[36] CVE-2014-2783. Common Vulnerabilities and Exposures [Online]. Available: http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2783
[37] CVE-2014-2001. Common Vulnerabilities and Exposures [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2014-2001
[38] CVE-2013-7373. Common Vulnerabilities and Exposures [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-7373
[39] CVE-2013-6951. Common Vulnerabilities and Exposures [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-6951
[40] CVE-2013-6950. Common Vulnerabilities and Exposures [Online]. Available: http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-6950
[41] G. Becker, F. Regazzoni, C. Paar, and W. P. Burleson, "Stealthy dopant-level hardware trojans," in *Proc. 15th Int. Conf. Cryptographic Hardware Embedded Syst.*, 2013, pp. 197–214.
[42] (2015). Nest Labs. Open source compliance [Online]. https://nest.com/legal/compliance
[43] Texas Instruments, "AM3715, AM3703 Sitara ARM Microprocessor," 2011.
[44] D. Halperin, T. Heydt-Benjamin, B. Ransford, S. Clark, B. Defend, W. Morgan, K. Fu, T. Kohno, and W. Maisel, "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses," in *Proc. IEEE Symp. Security Privacy*, 2008, pp. 129–142.
[45] ST Microelectronics, "STM32L15xQC, STM32L15xRC-A, STM32L 15xVC-A, STM32L15xZC Ultra-low-power 32b MCU ARM-based Cortex-M3, 256KB Flash 32KB SRAM, 8KB EEPROM, LCD, USB, ADC, DAC," no. 026119 Rev 5, 2015.
[46] M. BARBARO and T. Zeller Jr., "A face is exposed for aol searcher no. 4417749," *The New York Times*, 2006.
[47] I. Reynolds and C. Fujioka. (2011). Update 2-sony removes data posted by hackers, delays playstation restart. *Reuters* [Online]. http://www.reuters.com/article/2011/05/07/sony-idUSL3E7G701T20110507
[48] Z. Whittaker. (2012). Amazon's zappos in massive data breach 24 million affected. *ZDNet* [Online]. http://www.zdnet.com/article/amazons-zappos-in-massive-data-breach-24-million-affected/
[49] M. Mombrea. (2014). Googles real plan behind the purchase of the nest thermostat [Online]. http://www.itworld.com/consumerization-it/416110/googles-plan-rake-cash-nest-thermostat
[50] W. Arbaugh, D. Farber, and J. Smith, "A secure and reliable bootstrap architecture," in *Proc. IEEE Symp. Security Privacy*, 1997, pp. 65–71.
[51] Texas Instruments, "MSP430 Programming Via the JTAG Interface," 2015.



Orlando Arias is a senior computer engineering student at the University of Central Florida, where he is currently a research assistant in the Security in Silicon Laboratory. His research interests include device security, secure computer architectures, network security, IP core design, and integration and cryptosystems. He received the Best Paper Award at the 52nd Design Automation Conference as part of his work in hardware-assisted control flow integrity systems. He is a student member of the IEEE.



Jacob Wurm is currently a senior undergraduate student studying computer engineering at the University of Central Florida. He is currently a research assistant in the Security in Silicon Laboratory lead by Dr. Yier Jin. His research interests include embedded device security, secure communication protocols, and network traffic analysis. He is a student member of the IEEE.

Khoa Hoang is an undergraduate student at the University of Central Florida. He enjoys tinkering with electronics, technology, and jail breaking devices. He has disclosed exploits of various Internet of Things (IoT) device and other smart devices to multiple vendors. He is currently listed on multiple "Security Hall of Fame" pages for successful bug bounty submissions.

Yier Jin received the BS and MS degrees in electrical engineering from Zhejiang University, China, in 2005 and 2007, respectively, and the PhD degree in electrical engineering in 2012 from Yale University. He is currently an assistant professor in the EECS Department, University of Central Florida. His research focuses on the areas of trusted embedded systems, trusted hardware intellectual property (IP) cores, and hardware-software co-protection on computer systems. He proposed various approaches in the area of hardware security, including the hardware Trojan detection methodology relying on local side-channel information, the post-deployment hardware trust assessment framework, and the proof-carrying hardware IP protection scheme. He is also interested in the security analysis on Internet of Things (IoT) and wearable devices with particular emphasis on information integrity and privacy protection in the IoT era. He received the Best Paper Award at the 52nd Design Automation Conference in 2015. He is a member of the IEEE.