Security Analysis and Exploitation of Arduino devices in the Internet of Things

Carlos Alberca*
MSc in Cybersecurity
Universidad Carlos III de Madrid, Spain
carlos.alberca@alumnos.uc3m.es

Sergio Pastrana
COSEC Research group
Universidad Carlos III de Madrid, Spain
spastran@inf.uc3m.es

Guillermo Suarez-Tangil†
COSEC Research group
Universidad Carlos III de Madrid, Spain
guillermo.suarez.tangil@uc3m.es

Paolo Palmieri
Department of Computing and Informatics
Bournemouth University, UK
ppalmieri@bournemouth.ac.uk

ABSTRACT

The pervasive presence of interconnected objects enables new communication paradigms where devices can easily reach each other while interacting within their environment. The so-called Internet of Things (IoT) represents the integration of several computing and communications systems aiming at facilitating the interaction between these devices. Arduino is one of the most popular platforms used to prototype new IoT devices due to its open, flexible and easy-to-use architecture. Ardunio Yun is a dual board microcontroller that supports a Linux distribution and it is currently one of the most versatile and powerful Arduino systems. This feature positions Arduino Yun as a popular platform for developers, but it also introduces unique infection vectors from the security viewpoint. In this work, we present a security analysis of Arduino Yun. We show that Arduino Yun is vulnerable to a number of attacks and we implement a proof of concept capable of exploiting some of them.

1. INTRODUCTION

The Internet of Things (IoT) is a fundamental paradigm of modern computing. Originally, the Internet was designed as a network interconnecting computers. But as objects – devices, appliances, vehicles, buildings. . . – become increasingly smart (that is, capable of computational tasks), the Internet is being populated by "things", rather than actual computers [7]. From home automation systems (such as smart alarms or smart home appliances) to medical devices (such as pacemakers or insulin pumps), smart objects are now able to interact with each other in an autonomous manner. Networks of sensors, for instance, can independently collect and exchange data, and trigger actions based on an analysis of the information. For example, weather sensors can collect data on rain in real time, and automatically activate appropriate road signals in case of a storm, all without human intervention [6].

While a great number of different platforms designed for the Internet of Things are currently available, Arduino is arguably one of the most popular [4]. Arduino is a flexible micro-controller and development environment, that can be used to control devices and read data from all kinds of sensors, and is easily embedded into existing applications. Due to its open source nature, inexpensiveness and versatility, Arduino provides an easy implementation framework [4]. Early adopters contributed by writing documentation and distributing the software they developed under open source licenses or under public domain. This resulted in a large open community, which makes it easier for new users to learn the platform and start developing and creating applications.

However, while the Arduino business model made it a popular choice among the non experienced users, the security of the platform has not attracted sufficient scrutiny. In fact, compared to other common IoT components (such as popular RFID tags), the security of systems based on Arduino has never been thoroughly analyzed. Indeed, it is not clear to what extent and adversary can profit from common exploitation techniques such as buffer overflows or code reuse attacks [5,8] in Arduino devices with connection capabilities.

In this paper, we focus on one of the most common Arduino devices, the Arduino Yun. One of the distinguishing characteristic of Yun is that it is composed by a two-tier architecture. In a lower level, it contains a classical Atmel AVR chip (shared by other Arduino devices). At a higher level, instead, the board includes an a Atheros processor supporting Linino, which is a Linux distribution based on OpenWrt. The two components are connected by a bridge.

In this paper, we provide an analysis of the internal Arduino Yun architecture and its components from a security perspective. We reverse both major components of Arduino Yun, i.e.: the Linux environment and the Arduino environment. We find that Arduino does not provide any memory

*This work was done as part of the MSc project of the *Master in Cybersecurity* at Universidad Carlos III de Madrid.

†Currently at Royal Holloway University of London, guillermo.suarez-tangil@rhul.ac.uk.

© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

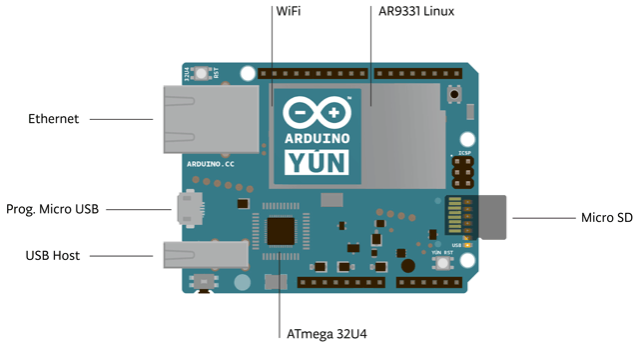

Figure 1: Arduino Yun [1]

failure control or other equivalent security measures, such as Address Space Layout Randomization (ASLR) or Canary Stack Protection. Based on that, we show how Arduino is vulnerable to attacks by providing a proof of concept attack, exploiting a vulnerability of the lower-layer architecture, based on Arduino, to gain privileges in the higher-layer chip, based on Linux. More precisely, we show how to exfiltrate credentials, perform DoS attacks, perform update attacks, and install a rootkit.

2. THE ARDUINO YUN PLATFORM

Arduino is an open-source platform providing "easy-to-use hardware and software intended for anyone making interactive projects" [1]. The Arduino platform was originally aimed at small electronics and micro-controller projects. However, with the increasing interest in IoT, a new board, the Arduino Yun, was specifically designed for IoT applications. The board combines the low-level electronics originally present in other Arduino devices with higher level architectures running a Linux based operating system. This section provides a description of the characteristics and architecture of Arduino Yun, with a focus on the design choices and their impact on security.

The Arduino Yun is composed of one micro-controller board based on two separate chips. The first, lower level chip is the ATmega32u4, while the second, higher-level chip is an Atheros AR9331 (Figure 1). The Atheros processor runs a Linux distribution based on OpenWrt named OpenWrt-Yun. Among others, the board features a built-in Ethernet and WiFi support (which provides the networking capabilities), a USB-A port, and a micro-SD card slot.

The main difference of Arduino Yun with other Arduino boards in that it integrates an on-board Linux distribution. Thus, it provides advanced communication capabilities and offers a powerful networked computer (Figure 2). In addition to Linux commands like cURL, developers can write their own shell and python scripts for robust interactions. Both parts, the Arduino and Linux environments, can connect to each other through a *bridge*. The bridge is a logical component programmed in python that contains different modules (including `bridge.py`, `packet.py`, `mailbox.py`, `processes.py`). The bridge facilitates communication, giving Arduino sketches the ability to run shell scripts, communicate with network interfaces, and receive information from the AR9331 processor. Consequently, this is a critical component from a security perspective: as we show in

Table 1: Specification of the AVR Arduino microcontroller and the Atheros microprocessor

Chip	AVR ATmega32u4 8bits	Atheros AR9331
Voltage	5V	3.3V
Flash Memory	32 KB (4 KB bootloader)	16 MB
SRAM	2.5 KB	2.5 KB
EEPROM	1 KB	1 KB
Clock Speed	16 MHz	16 MHz

the following, by exploiting Arduino environment it is possible to bypass inner security mechanisms, therefore compromising the Linux environment. Moreover, as we show during this work, the bridge requires no authentication in the Linux environment, and all the commands issued from the ATmega32u4 chip are executed with root privileges.

Table 1 provides details of the hardware specifications for both the ATmega32u4 and the Atheros chips. Both chips are clearly resource constrained, compared to classical computer boards. However, it is noteworthy to observe the difference in processing capabilities of both chips (for instance, 2.5KB of RAM are available in ATmega32u4, compared to the 64MB in the Atheros chip). The amount of RAM memory available in the Atheros chip makes it suitable to run a lightweight operating system like OpenWrt.

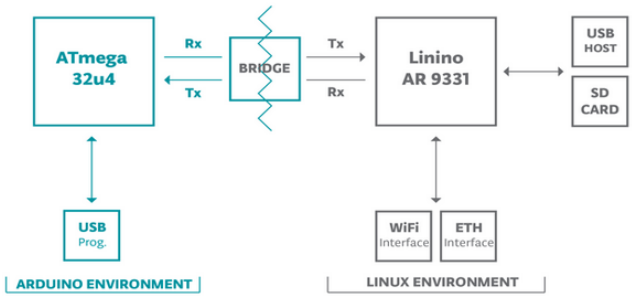

Figure 2: Arduino Yun Diagram [1]

For the purpose of our analysis, we have used the latest stable image `OpenWrt-Yun 1.5.3`, released on November 13th, 2014.

2.1 Linux Environment

The Linux Environment of Arduino Yun is based on a dedicated distribution derived from OpenWrt, which includes the Linux kernel v3.3.8. The Arduino Yun OpenWrt comes with some packages installed, including Dropbear sshd v2011.54, which provides the functionality of an SSH server (required for remote administration). In the next section, we show that some of these packages may expose the system to vulnerabilities if not properly patched.

Should an attacker gain unlimited access to the filesystem, the system could be easily compromised. In particular, we identify the following paths as main targets:

- `/var/run` → information about processes and pidfiles.
- `/var/run/wpa_supplicant-wlan0.conf` → WiFi parameters stored in cleartext.
- `/tmp/resolv.conf.auto` → DNS servers addresses.

- `/etc/arduino` → GPG keys used for verifying packages.
- `/etc/config` → general configurations (Arduino, ssh, firewall, wireless, etc.). In particular, information includes WiFi keys, the REST API security flag (which can be set to to 'false' to disable password checks), ssh authentication, and so on. The Arduino access password is stored using SHA256 hash function.
- `/etc/dropbear` → SSH RSA keys.
- `/etc/hosts` → mapping of hostnames to IP addresses.
- `/etc/avrdude.conf` → pin mapping and other parameters that are used by avrdude for the ATmega32u4.
- `/etc/opkg.conf` → repo used for package updates.
- `/sys/` → modules for crypto, drivers, firmware, etc.
- `/rom/` → if files are changed and stored here, the changes are maintained even after a factory reset.
- `/usr/bin/kill-bridge` → the script used to kill the bridge service between the Arduino and Linux environment. Another script (`run-bridge`) can be run to start again the bridge, after changes are made to `bridge.py`.
- `/usr/bin/wifi-reset-and-reboot` → used to reset wifi parameters and reboot the system.
- `/usr/lib/python2.7/bridge/` → python bridge scripts that define the functionality of the bridge (including `bridge.py`, `mailbox.py`, `processes.py`).

2.2 Arduino Environment

The Arduino environment contains an ATmega32u4 chip based on AVR architecture, which is a modified Harvard Architecture [5]. It has 32 registers and the memory is divided into three types: EEPROM (as a HDD), SRAM (is dynamic in the execution time) and Flash (where code is loaded). Figure 3 shows an schematic view of the most relevant memory areas of the SRAM and its base addresses (i.e., the registers, heap and stack). We consider these areas as they are classical targets of software exploitation and other attacks.

The *32 registers* occupy positions from *R0* (0x00) to *R31* (0x1F). The registers *R26* to *R31* have some added functionalities that distinguish them from the general-purpose registers preceding them. These registers are 16-bit address pointers used for indirect addressing of the data space. More precisely, these registers conform 16-bit indirect address pointers as follows: X=0x[*R27*:*R26*], Y=0x[*R29*:*R28*], and Z=0x[*R31*:*R30*].

The *data area* contains global variables used by the program that are not initiated to zero. The *BSS* segment contains all global variables that are initiated to zero and constant strings. The *AVR Stack Pointer (SP)* points to top of the stack (that is, the data SRAM Stack area) where the Subroutine and Interrupt Stacks are located; it is implemented as two 8-bit registers (*SPH*:*SPL*) in the I/O space. The *stack*, that grows from higher to lower memory locations, contains the return addresses for subroutines, saved registers and local variables, and when it grows, the deallocation process is automatic. Nevertheless, the *heap*, that grows from lower to higher memory locations, is managed by the functions `malloc()`, `realloc()` and `free()`. It is shared by all common libraries and dynamically loaded modules in a process, and must be deallocated manually (using `free()`).

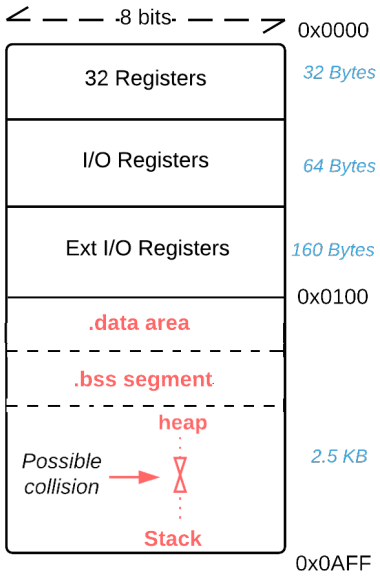

Figure 3: ATmega32u4 SRAM memory

Arduino Yun offers an open source integrated development environment, the Arduino IDE. The environment is written in Java and based on Processing (an open-source library), and provides the tools to integrate Arduino in alarms, boiler controllers, home controllers and similar applications.

Arduino Yun also offers a remote administration panel. By default, the board WiFi adapter is configured in access point (AP) mode, and advertises a WiFi hotspot (ARDUINO-YUN90XXXXXX). By connecting to this hotspot, the administration website is accessible through a browser at IP address 192.168.240.1. From the interface it is possible to configure the Yun to connect to a specific wireless network in client mode. Arduino reverts to AP mode after losing connection to the configured host WiFi for one minute.

3. SECURITY ANALYSIS

In this section we provide a security analysis of the Arduino Yun platform. We describe how an adversary can compromise the device by exploiting vulnerabilities found in its design, both at the Arduino and Linux level.

3.1 Linux Environment

We first analyze the Linux environment from an external attacker point of view, using two popular open-source tools, Nmap[1] and Nessus[2], from a remote host. The following vulnerabilities have been found in the default software packages (with the respective CVE identifier):

- Dropbear sshd v2011.54: the remote SSH service is affected by multiple vulnerabilities that allows an attacker to perform three different attacks. First, it is possible to run a Denial of Service (DoS) against the

[1] www.nmap.org/
[2] www.tenable.com/products/nessus-vulnerability-scanner

server, provoked by the way the `buf_decompress()` function handles compressed files (CVE-2013-4421). Second, it is possible to perform a User-enumeration due to a timing error when authenticating users (CVE-2013-4434). Finally, and most critically, adversaries can execute remote code (CVE-2012-0920) when the so-called `use-after-free` condition runs on concurrency channels.

- BusyBox v1.19.4. The DHCP client implementation (`udhcpc`) allows malicious remote servers to execute arbitrary commands via shell script meta-characters (CVE-2011-2716).
- Dnsmasq v2.62. The DNS server may respond from some prohibited interfaces, allowing an attacker to run a DoS attack (CVE-2013-0198 and CVE-2012-3411).
- Linux Kernel v3.3.8. The kernel version shipped with the device is outdated, and affected by a number known vulnerabilities. In particular, a firewall (`netfilter`) bug allows an attacker to crash the system by performing a DoS attack (CVE-2014-2523). A local user privilege escalation vulnerability is also present (CVE-2013-1763). However, we note that available exploits for these vulnerabilities would need to be adapted to the Arduino Yun platform, as the kernel has been compiled specifically to run on embedded devices.

While all these vulnerabilities have related patches and updates, the official Arduino repository[3] has not been updated at the time of writing. Users would therefore need to apply the patches manually.

We continue the security analysis by analyzing the bridge interface. In particular, we note that there are no integrity checks on critical files, including python scripts (such as `bridge.py`) or binaries. It is therefore possible for an attacker to replace or tamper with the files. While the original files can be recovered by performing a factory reset, this can be circumvented by storing a copy of modified files in the `/rom/` directory (as explained in Section 2.1). Moreover, permission on the scripts `kill-bridge` and `wifi-reset-and-reboot` allow a local Denial of Service attack to be performed on the bridge, thus breaking the connection between the Arduino and Linux environments.

An analysis of the network configuration reveals further vulnerabilities. The Arduino administration interface can be accessed via port number 80 (HTTP), without being redirected to port 443 (HTTPS). Therefore, a normal connection can be sniffed from a node within the same local network. The default behavior of reverting to an open access point mode after the wireless interface is unable to connect to the configured hotspot for one minute also opens the way to deauthentication attacks, that can force the Yun into a malicious network. By sending disassociate packets to the Yun, we can also recover a hidden ESSID, or capture WPA/WPA2 handshakes by forcing clients to reauthenticate.

3.2 Arduino Environment

The analysis of vulnerabilities for the Arduino environment is not as straightforward as in the case of Linux, and no automated tools are available for vulnerability scanning. Therefore, we have conducted the vulnerability search by performing a static analysis of the memory address map, following a "trial-and-error" approach. Additionally, unlike other architectures such as Intel x86 or ARM, the Atmel32u4 lacks debugging tools.

The Arduino IDE compiler (i.e. avr-gcc version 4.8.1) does not perform any security checks during compilation. Therefore, when a sketch (program) runs out of memory, no warning nor segment violation error is issued. We therefore try to exploit out-of-memory errors. We check errors in two different parts of the SRAM memory: the heap and the stack.

- In the case of the heap, we were able to produce a Heap Buffer Overflow (see Figure 4), by allocating consecutive memory buffers in the heap (whose corresponding variables are named `cmd` and `arg`) using the `malloc()` function. The overflow occurs when overwriting the first set of data (i.e. `cmd`) with data from the second set (i.e. `arg`). In Section 4 we provide further details of this exploitation technique.

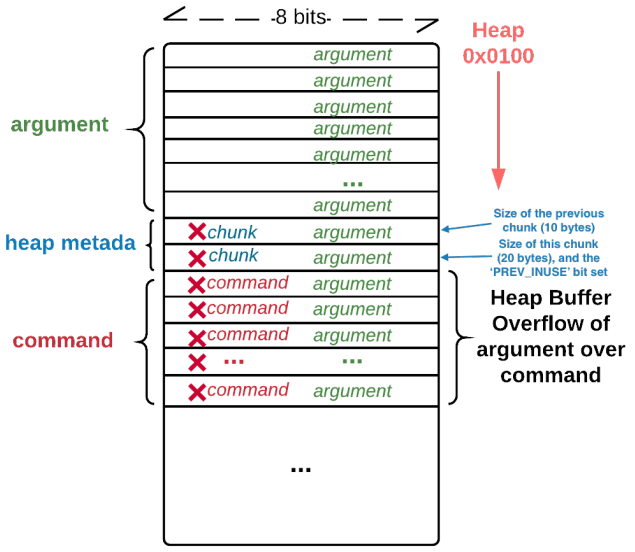

Figure 4: ATmega32u4 Heap Buffer Overflow

- In case of the stack, it is also possible to perform a Stack Buffer Overflow attack. The lack of debugging tools hardens us achieving control of the program flow. Moreover, due to the physical separation of SRAM (which is non executable) and Flash memory (where the code resides), it is impossible for an adversary to execute shellcode. Thus, only attacks based on code reuse attacks such as Return Oriented Programming are feasible, as showed first by Francillon and Castellucia [5] and recently by Habibi et al. [8]. Additionally, due to the low size of the internal memories, we note that the heap and the stack can collide when a big amount of subroutine calls with many local variables are performed (see Figure 3).

4. PROOF OF CONCEPT ATTACK

In this section, we present two practical attacks that can be performed on the Arduino Yun IoT platform. The attacks demonstrate the vulnerabilities found in the security

[3]http://downloads.arduino.cc/openwrtyun/1/packages/

analysis presented in the previous section. One of the attacks has been fully implemented, and the code of a proof of concept vulnerable program that can be used as attack vector is publicly available[4].

In the attacks, we exploit the two-layer nature of Arduino Yun: we target the lower-level Arduino environment, which our analysis revealed to be the most vulnerable component, in order to obtain results at the higher-level Linux environment. We achieve this by exploiting the lack of memory security checks in place in Arduino, and we utilize the bridge component linking the two layers to execute code on Linux. The purpose of the bridge component is in fact to allow Arduino-based software to run Linux commands: we will therefore need a vulnerable software utilizing the bridge to use as attack vector.

4.1 ATmega32u4 Heap Buffer Overflow

For the purpose of demonstrating the attack, we developed (with Arduino IDE) a sample program that uses the bridge functionality of the board. The functionality of the specific program is not important for the success of the attack, as long as it includes Linux commands to be executed through the bridge interface, therefore allowing us to use it as attack vector. The source code of the program is provided and available for download from our web site[4].

The sample program we developed can be used to configure the network interfaces of Arduino Yun via bluetooth (this could be helpful, for instance, in cases when direct connections to those interfaces are impossible due to firewall configuration or other reasons). Bluetooth is used to ask and receive data from users, and the collected data is stored in a variable (named `arg`). The data would normally correspond to the arguments required to configure the network interfaces. The program calls a static command (`ifconfig`, hardcoded into the source code) in order to configure the network interfaces, executing it through the bridge. We also implement a security check that prevents the user from executing more than one command, by checking the presence of & or ` characters in the `arg` buffer. The two data buffers (10 Bytes for `arg` and 20 Bytes for `cmd`) are assigned on the heap with `malloc()`. `arg` is used to store input supplied by the user by bluetooth, copied with `strcpy()`.

Given the vulnerable program described, we can perform a buffer overflow attack on the heap. The goal of the attack is to overwrite the `ifconfig` command with a different command. This attack is particularly effective due to the Arduino Yun architecture: as the `cmd` variable is executed in the Linux environment through the python-based bridge, it is executed with root privileges by default. In order to perform the attack, the following malicious data has be stored as `arg`, causing the buffer of *cmd* to be overwritten (see Figure 4):

- 10 bytes of data (overwriting `arg`) +
- 2 additional bytes (overwriting the heap metadata) +
- the command we wish to execute maliciously.

We successfully tested the following malicious payloads:

- `12 blank data + cp /etc/shadow /www/proof.txt`
  Credential exfiltration: copies the content of the file shadow in the folder www, so it can be retrieved by an HTTP request.
- `12 blank data + /usr/bin/kill-bridge`
  DoS attack: kills the bridge between the two chips, causing a service interruption.
- `12 blank data + echo 0 >/proc/sys/kernel/randomize_va_space`
  Downgrade attacks: disables address space randomization in the Linux environment.
- `12 blank data + opkg sslstrip`
  Payload injection: installs a malicious package to perform attacks on SSL.
- `12 blank data + wget http://myserver.com/rootkit.sh && ./rootkit.sh`
  Persistence attacks: downloads and executes a script, which could be for example a rootkit.
- `12 blank data + cd /etc/ && wget http://myserver.com/opkg.conf`
  Update attacks: replaces the configuration file where the package repositories are downloaded.

4.2 ATmega32u4 Stack Buffer Overflow

Using a similar strategy to that of the attack above, we performed a stack buffer overflow in the Arduino environment. In order to do so, we developed a program that first reads an input from the serial interface in `main()`, and then passes this input to another buffer in the function `f1()`. The original input can have a maximum length of 30 bytes, but we set the buffer in `f1()` to be only 10 bytes long. Because `strcpy()` does not check memory boundaries, a buffer overflow will occur. If the program is using the bridge to send data from the Arduino environment to the Linux one, we can cause the Linux level to crash on inputs from the Arduino level, thus achieving a denial of service. While the lack of debugging tools for Arduino, prevented us from gaining the understanding of the memory management, we noted that the program crashes when specific payloads where sent.

In particular, we used a black block approach based on *trial and error*. More precisely, after running the buffer overflow attack, we observed that the return address of the stack was overwritten. We further observed that the compiler integrated in Arduino IDE (avr-gcc version 4.8.1) does not provide countermeasures against memory corruption vulnerabilities. **Address Space Layout Randomization**, ASLR [3] and **Stack Canaries** [2] are two common countermeasures that obfuscate the memory layout of the targeted code and prevent stack overflows respectively. Thus, and adversary can carefully send specific payloads to gain the control flow of the program and reuse code from the program memory [5, 8].

5. DISCUSSION

Our results expose a general lack of security in the design of the main Arduino Yun components, and highlight the urgent need to enhance security at both layers: the Atmel chip and the Linux OpenWrt distribution. The Atmel ATmega32u4 running the Arduino environment proved to be vulnerable to memory attacks. Due to the limited resources of the technology, full-fledged memory checks may not be implementable: however, we believe that lightweight memory

[4]www.seg.inf.uc3m.es/~spastran/mal-iot-sample

controls in the Atmel chip should be a requirement for any future board of the Arduino family. For example, recently Habibi et al. have proposed an memory address obfuscation technique on Unmanned Aerial Vehicles (UAVs) [8], using additional hardware. Since this solution seems promising, it is not clear to what extent the costs of adding new hardware are affordable in real settings, where the physical space and resources available are limited.

In absence of this, security cautions must be taken by the Arduino developers, which we believe that currently are not aware of the security breaches they might inadvertently leave in their programs. In order to help developers design robust and secure programs, it would be desirable to have a set of guidelines from the Arduino community, similar to the "Best practices for Security and Privacy" published for Android[5]. Such guidelines would help to develop secure programs, and could include buffer handling, free memory functions and stack and heap collisions, input/output control on serial port, pin layout, etc. Further warnings and errors could also be implemented in the Arduino IDE compiler, with a particular focus on memory protection.

Given that many Arduino devices may have direct connection to the Internet, thus exposing them to remote attacks, the Linux OpenWrt distribution should be updated more often, in order to include all the latest security patches and updates. This is currently not the case, as shown in Section 3.1. Using specialized vulnerability search engines such as Google Hacking and Shodan.io, that crawl the web looking for vulnerable devices exposed to the Internet, we found a significant number of vulnerable Arduino Yun boards accessible remotely.

At the Linux environment level, it is clear that `root` should not be the only and default user in the system, especially when executing programs through the bridge interface. The set up of a specific user with limited permissions to handle commands from the non-secure Atmel chip could be a potential solution, with another user to handle remote ssh connections. It would also be beneficial for the security of the system to implement integrity controls to detect changes on critical files on the filesystem. Due to the limited resources of the platform, providing an exception handling control (such as try/catch sentences), to keep the system running when a stack buffer overflow occurs would be a challenge on Arduino and more generally in any IoT environment, but we believe this could be an interesting research direction to explore in the future.

6. CONCLUSION

The Internet of Things offers almost unlimited opportunities for smart applications, and is one of the main drivers of modern computing. However, as the use of IoT enabled devices increases, so does the attack surface area, causing new vulnerabilities to be discovered (and in some cases exploited) routinely. The low-cost, low-power nature of many IoT platforms also means that security measures that have been standard in computers for years are often missing in IoT boards. Given the critical nature of many IoT applications scenarios, including ubiquitous monitoring and health services, the security of IoT systems and solutions should be a major priority for all relevant stakeholders.

[5]https://developer.android.com/training/best-security.html

In this paper we provide a security analysis of one of the most popular devices used in IoT applications, the Arduino Yun. In particular, we analyzed the two-layer internal architecture of the device (with a lower layer running an Arduino environment and a upper layer running Linux), and found how a lack of security checks in the lower layer can cause the system to be compromised at all levels. In order to show this, we presented a proof-of-concept attack over a vulnerable application, which we have developed for this purpose. By exploiting the interface that interconnects the two layers, called *bridge*, we have been able to perform a series of more serious post-exploitation attacks, such as rootkit installation and opening backdoors. The attacks are fully practical, and can be applied to software developed by the Arduino community.

Acknowledgment

This work was partially supported by the MINECO Grant TIN2013-46469-R (SPINY: Security and Privacy in the Internet of You) and by the CAM Grant S2013/ICE- 3095 (CIBERDINE). An extended work of this paper was presented as a MSc Thesis in Master in Cybersecurity at University Carlos III de Madrid.

7. REFERENCES

[1] M. Banzi, D. Cuartielles, T. Igoe, G. Martino, and D. Mellis. Arduino official. http://www.arduino.cc.

[2] A. Baratloo, N. Singh, T. K. Tsai, et al. Transparent run-time defense against stack-smashing attacks. In *USENIX*, pages 251–262, 2000.

[3] S. Bhatkar, D. C. DuVarney, and R. Sekar. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. In *USENIX Security*, pages 105–120, 2003.

[4] C. Doukas. *Building Internet of Things with the Arduino*. CreateSpace Independent Publishing Platform, USA, 2012.

[5] A. Francillon and C. Castelluccia. Code injection attacks on harvard-architecture devices. In *ACM CCS 2008*, pages 15–26. ACM, 2008.

[6] S. Gaitan, L. Calderoni, P. Palmieri, M.-C. Ten Veldhuis, D. Maio, and M. van Riemsdijk. From sensing to action: Quick and reliable access to information in cities vulnerable to heavy rain. *Sensors Journal, IEEE*, 14(12):4175–4184, Dec 2014.

[7] N. Gershenfeld, R. Krikorian, and D. Cohen. The internet of things. *Scientific American*, 291(4):46–51, 2004.

[8] J. Habibi, A. Gupta, S. Carlsony, A. Panicker, and E. Bertino. Mavr: Code reuse stealthy attacks and mitigation on unmanned aerial vehicles. In *IEEE ICDCS 2015*, pages 642–652. IEEE, 2015.