ArduWorm: A Functional Malware Targeting Arduino Devices

Sergio Pastrana, Jorge Rodriguez-Canseco, Alejandro Calleja
Computer Security Lab. Universidad Carlos III de Madrid
Avda. Universidad 30, 28911, Leganes, Spain
{spastran,jorrodri,accortin}@inf.uc3m.es

***Abstract*—The Internet of Things (IoT) is a growing market which provides several benefits for industry, governments and end users. However, the increasing use of embedded and pervasive devices introduces new vulnerabilities in the network. In the last years, the number of malware and exploits targeting the IoT has grown considerably, which issues a challenge for the industry and the academy. To further motivate this challenge, in this paper we describe a malware piece targeting Arduino Yun, which is a common platform used in IoT scenarios. The malware, dubbed ArduWorm, is able to bypass all the security implemented in Arduino by exploiting a memory corruption vulnerability and hijack the device. Moreover, due to the architectural flaws found in Arduino Yun, the malware is able to get the control of a Linux-based microprocessor integrated in the device with full privileges, which allows it to install a backdoor and spread as a worm through the compromised network.**

***Index Terms*—Malware, Internet of Things, Arduino, Worm, Remote Access Tool**

Type of contribution: *Research in progress*

I. Introduction

The Internet of Things (IoT) comprises the set of technologies, networks and architectures that allow the connection of different embedded and pervasive devices, like wearables, sensors or smart phones. It provides several facilities and economical benefits to end users, companies and organizations. Nowadays, the IoT is an emerging topic in both the industry and the academy, and it is expected that the number of devices fabricated and used in IoT projects will exceed the 50 billions by 2020[1]. IoT projects are present in diverse scenarios, including smart cities, automotive industry or Body Area Networks (BAN), to name a few.

In the last few years, the IoT has gained the attention of malicious actors due to several reasons. Modern smart devices such as smart phones or tablets have substituted PCs and laptops for a plethora of user applications like social networking, instant messaging or e-commerce. Therefore, these devices store a huge amount of personal and valuable information that is attractive for attackers. Moreover, the connection capabilities of IoT devices and the open scenario where they are usually deployed offer new infection vectors to potential adversaries. Indeed, since these devices are usually connected between them (e.g. forming Wireless Sensors Networks) and/or to the Internet, they are attractive targets for botnets, and recently the concept of "Thing-Bot" (i.e., a compromised device used as part of a large botnet) has gained popularity [1], [2].

One common platform used in embedded devices is Arduino. Arduino is an open-source platform providing "easy-to-use hardware and software intended for anyone making interactive projects" [2]. The Arduino platform was originally aimed at small electronics and micro-controller projects. Particularly, with the increasing interest in IoT, a new board, the Arduino Yun, was specifically designed for IoT applications. Together with the classical Atmel AVR based Micro Controller Unit (MCU) present in most of Arduino devices (concretely, the ATmega32u4), the Yun is also equipped with a Atheros Micro Processor (MPU) holding a Linux based OpenWrt operating system. This Atheros MPU manages one Ethernet interface and one Wifi card, which makes it a suitable device for IoT scenarios. Both the Atmel AVR and the Atheros are connected using a serial bus managed by a software library called *Bridge*.

A security analysis of the Arduino Yun shows that it contains many architectural flaws. We have evaluated the attack surface of this device, and found a critical point of exposure in the *Bridge* library connecting both chips that allows to compromise the entire device by exploiting a memory corruption vulnerability in the Atmega32u4 MCU. Moreover, since this AVR-based chip has limited resources compared with modern MCU and MPU based on ARM or x86 architectures, classical protections against memory corruption, such as stack overflow protection or memory layout randomization can not be easily deployed.

In this work, we present ArduWorm, a proof-of-concept malware that targets Arduino Yun devices. The exploitation of a vulnerability in the Atmel MCU allows ArduWorm to bypass the *Bridge* and compromise the OpenWrt to establish a backdoor. AVR is based on a modified Harvard architecture where the code and data memories are physically separated, thus hindering code injection. Thus, the exploit uses code reuse attacks (i.e., Return Oriented Programming and return-to-lib) to benefit from a memory corruption vulnerability. ArduWorm has reconnaissance and infection capabilities, and it can automatically spread through neighbor nodes. While this specimen is just a proof of concept proven in our experimental setup, we hope that it can motivate research in the design of defensive mechanisms for Arduino devices.

This paper is structured as follows. In Section II we review the current state of the art in malware specific for the IoT and provide some useful background. In Section III we present the exploitation mechanism implemented, and in Section IV the details of ArduWorm. Finally, in Section V we discuss about possible countermeasures and in Section VI we present the conclusions of the work.

[1] http://iot.ieee.org/newsletter/january-2016/the-rise-of-iot-why-today.html

[2] From the Arduino official website, http://www.Arduino.cc

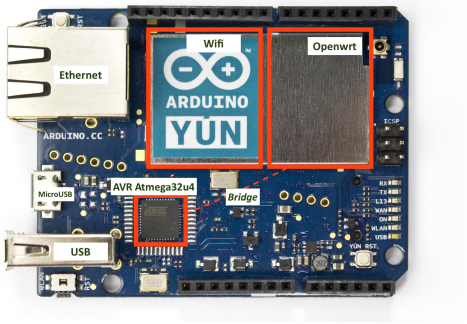

Fig. 1. Arduino Yun

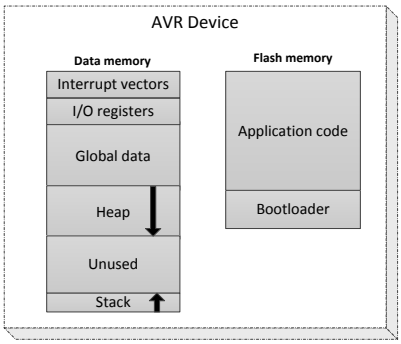

Fig. 2. Schematic view of AVR memories

II. Background and Related Work

This section first describes the targeted system, i.e. the Arduino Yun, along with the AVR architecture. Then, we overview memory corruption attacks, which form the basis of the initial exploitation used by ArduWorm. Finally, we review current state of the art and provide some examples of malware targeting IoT devices from different scenarios.

A. Arduino Yun

Arduino Yun is different from other Arduino boards because it includes a preinstalled Linux distribution, and integrates advanced communication capabilities, thus offering a powerful networked embedded system with the ease of Arduino. Concretely, the Arduino Yun is composed of one micro-controller board based on two separate chips, i.e. the Atmel AVR ATmega32u4 and an Atheros AR9331 running the OpenWrt

The two chips are connected through a serial bus managed by a *Bridge* library, which is included as part of the Arduino Library and is implemented in Python in OpenWrt. The *Bridge* provides Arduino sketches with the capacity of running 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 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.

The integrated Atmega32u4 chip is based on the AVR architecture. AVR is a modified Harvard architecture implemented by Atmel in 1996. It stores code and data in memory chips that are physically separated, i.e. the Flash memory and the data or SRAM memory (see Figure 2). The SRAM contains the program data, the heap and the stack. A property of AVR is that the stack starts at the highest address of the SRAM memory, and grows towards lower addresses (i.e., a *PUSH* instruction stores a new byte in the stack and decreases the stack pointer), while the heap grows towards higher addresses, which can lead to a heap/stack collision.

B. Memory corruption attacks

Memory corruption is probably one of the most exploited vulnerability so far. The adversary profits from an uncontrolled out-of-memory error to modify the memory map and hijack the control flow. For example, stack overflows allow the adversary to craft a specific payload that overwrites the return address (which is usually stored in the stack) and gain the control of the execution. Traditionally, the next step was to jump to an area of memory with code injected by the adversary (i.e. shellcode) intended to perform subsequent states in the attack (e.g. download additional code, open a reverse shell, etc.).

With the advance in defense mechanisms that prevent code injection, such as preventing memory to be both writable and executable ($W \oplus X$), code injection has become useless in modern systems. Thus, current exploitation techniques rely on reuse existing code from the program memory. An example of such code-reuse attacks is return-to-lib [3], that forces the program to execute code from imported libraries (e.g. libc), prior setting the desired arguments. This way, an attacker could execute arbitrary shell commands by invoking the *exec* function from libc and passing as arguments the string */bin/sh* together with the desired command.

A more sophisticated code-reuse attack is Return Oriented Programming (ROP), where the adversary forces the execution of different pieces of code (called *gadgets*) ending in a *ret* instruction. Thus, by carefully building a stack containing the addresses of these *gadgets*, the adversary can conform a *chain* that execute different pieces of code consecutively in order to perform the desired action. The reader may find more information about this attack in [4] or [5]. Moreover, in Section III we describe a code reuse attack that combines both ROP and return-to-lib in order to compromise the *Bridge* of Arduino Yun.

C. Malware in IoT

During the last few years, malware in tablets and smart phone devices has become one of the main concerns of security researchers. According to Mcafee's 2015 threat reports up to 1.2 million different malware pieces targeting mobile platforms were detected [6]. A similar report published by the AV company PandaLabs, stated that during 2015 an average of 230.000 different samples were detected on a daily basis [7].

An important novelty introduced by smart phones, is the capability of connecting other devices extending the functionality of the device. This is the case of several wearable devices like activity monitors or smart watches which are wirelessly connected to the smartphone conforming a so called personal area network. These devices count with sensing

capabilities such as sensors or GPS systems, making them sources of sensitive information and therefore appealing attack targets.

Recently, Symantec researchers have proven that ransomware installed in a smartphone can easily move to the paired wearables and infect them [8]. Ransomware is an emerging threat that allows attackers to obtain economical benefits directly from the infected users. It works by either preventing the access to the compromised system (e.g., by modifying the login credentials) or by hijacking the data stored (e.g. by encrypting its contents). In any case, the attacker claims for a ransom to free the stolen assets. Different ransomware pieces have been released for smartphones, like recent *Android.LockerPIN* which modifies the pin code of the infected phone.

Exploits and malware targeting embedded devices such as routers are not a new threat. However, the increasing proliferation of smart and pervasive devices in modern society opens a huge amount of new infection vectors. Indeed, IoT devices use embedded hardware and firmware that are resource constrained, often reused from other technologies. Thus, if an adversary is able to compromise one, she may be able to compromise several of them. Indeed, the cybersecurity company *Proofpoint* presented a media press in 2014 [2] incorporating the term *Thing-bot* to name the more than 100.000 compromised embedded devices that were used as part of a big botnet used to send spam and phishing. These attacks were issued by compromised devices such as smart TVs, fridges, routers, etc.

Since IoT devices are remotely accessible, mostly using protocols such as Telnet or SSH, remote exploitation has become the primary mean to compromise these systems. Pa et al. [1] analyzed the Telnet traffic to categorize the sources of the packets. They showed that most of the traffic was issued from devices such as DVR (Digital Video Recorders), IP Cameras or Wireless routers. Moreover, by using their proposed IOTPot (a honeypot focused on IoT threats) they captured and categorized malware for this kind of devices. They detected at least 43 new malware samples that targeted embedded architectures, and, even worse, only 4 of them were reported as malicious in VirusTotal.

Hernandez et al. [9] showed vulnerabilities in the boot process of the Google Smart Nest Thermostat, a device aimed at optimizing the air conditioning and heat consumption to save energy. This device is connected to the Internet and can be remotely configured. The firmware updates are signed and thus the manufactures claim that the device cannot be exploited, since signatures are verified prior to the installation of new updates. However, if an adversary has physical access to the device (e.g. during manufacture or transportation), then the boot process can be modified. They proposed a bootkit which installs a backdoor and avoids the signature verification. Moreover, since the device is typically connected to the home LAN, the authors showed how the Smart Nest could use ARP spoofing attacks to perform Man-In-The-Middle (MITM) attacks to any computer from this LAN.

SYNFul Knock [10] is a malware that affects firmware from some Cisco Routers. This malware profits from default or leaked credentials to install the malware and modify the firmware. This modified firmware provides the adversary with a backdoor (with *root* privileges) and listens for commands from a C&C server. Mandiant researchers have identified that these commands are intended to load additional modules to be run in the router. SYNFul Knock maintains persistence across reboots, and obfuscate the authentication with the C&C server by using special TCP packets during the initial triple handshake, by encapsulating the commands within non-standard TCP payloads (see the technical report published by Mandiant at [10] for further details).

Similarly, Linux Wifatch is a white-hat trojan [11] that installs a backdoor in home routers by exploiting a weak Telnet authentication. It installs a backdoor and is able to communicate with a C&C server. However, it is considered a white-hat piece of malware since it seems that no malicious action is performed, and even more, it tries to hinder infection by other malware pieces. Its source code (written in Perl) has been recently released and is open for the community[3].

An important concern in the IoT security is that the attacks can compromise not only the data and digital assets, like in classical PCs. Indeed, embedded systems may be physically implanted in humans, so if they are compromised, it can seriously harm the integrity of human beings. For example, in 2008 it was proven that pacemakers and cardiac defibrillators were vulnerable to radio frequency attacks [12]. Thus, security on Implantable Medical Devices (IMDs) pose an appealing area of research that requires efforts from both the academy and the industry [13].

The transport industry have additionally moved to the IoT. Nowadays, cars are equipped with a endless amount of sensors and electronic devices aiming at controlling the brakes, tires, or even providing Internet connectivity to the car. The research carried out by Miller and Valasek [14] (presented at Black Hat in 2015), showed how an adversary can control remotely the inner sensors of a car (e.g., the 2014 Jeep Cherokee was used in their proof of concept attack). This research demonstrates that attack surfaces for automobiles presented in prior works [15] are actually exploitable.

The analysis and study of current state of the art indicates that remote exploitation is by far the most used infection vector. It profits from default or weak passwords, vulnerabilities found in the device firmware or wrong configurations. Additionally, the constrained resources in many of the architectures used in these devices usually prohibits the implementation of defense mechanisms as antivirus or in-device firewalls. For example, embedded devices based on the AVR architecture, such as Arduino devices, lack of memory corruption controls like stack overflow protection or Address Space Layout Randomization (ASLR)[16]. Thus, it is possible to construct exploits that reuse existing code from the firmware, allowing malicious users to perform well-known Return Oriented Programming (ROP) or return-to-lib attacks. For example, Francillon and Castellucia [17] proved that using ROP they could inject code into the flash memory of an Atmel chip based on the AVR architecture. Recently, Habibi et al. [18] showed that a Unmanned Aerial Vehicle (UAV), most commonly refereed as *dron*, can be hijacked using the ROP to modify the flying gyroscope. In our work,

[3]https://gitlab.com/rav7teif/linux.wifatch

we extend these attacks and show that the entire device, including the OpenWrt, can be compromised by exploiting a vulnerability in the AVR chip. This allows to install further binaries or malicious scripts that allows it to gain persistence, install backdoors or automatically infect neighbor sensors, i.e. turning the malware into a worm.

III. Initial exploitation of Arduino Yun

In this section, we describe how an adversary can exploit the Arduino Yun using using ROP and calling functions from the *Bridge* library included in the main program. We assume that the adversary can somehow get the binary loaded at the targeted device (e.g., by physically dumping its content or getting it from public repositories). Moreover, since the stack in AVR is located at highest positions of the SRAM memory, the adversary has limited space for the exploitation payload. However, the adversary can inject larger payloads in memory by provoking software resets of the device and exploiting the vulnerability several times. We elaborate more on these assumptions in next subsection. Finally, we assume that the adversary can somehow gain the control of the program flow by remotely exploiting a memory corruption vulnerability on the device, namely a stack overflow. Next, we present the attack overview and provide implementation details.

A. Exploit overview

In this section we describe the initial exploitation used in ArduWorm. It is based on code reuse attacks. The main goal is to call a function from the Arduino library to run a command in the OpenWrt. However, the required function from Arduino libraries is *call by reference*, i.e.: arguments are passed as pointers to data memory. Indeed, different from other architectures, function arguments in AVR are passed via registers when possible and they are passed through the stack only when the arguments are larger than the number of available registers. Accordingly, the first step for the adversary is to inject data in the SRAM. We perform this step using Return Oriented Programming (ROP).

Injecting data in SRAM is limited by the size of memory available for the exploit. The main idea is to use ROP to store data into non-volatile areas of the SRAM memory [17], [18]. Concretely, we use a chain of gadgets (named *Store_data*) that allow an adversary to build a payload that load several values in memory recursively. We provide more details of these gadgets and how they are used in Section III-B.

The exploitation of a stack overflow has a limitation in AVR devices. The stack is always located at the highest address of the SRAM memory, and thus the space available to inject a payload after overflowing the stack is significantly limited. When a buffer is locally declared in a function, the return address is stored at a higher position of the memory reserved in the stack. This position may be close to the end of the SRAM address space (see Figure 2). Thus, the adversary may not be able to send large attack payloads as it is usually done in ROP attacks against conventional architectures [5]. To partially overcome this issue and provide more space, the stack pointer can be moved to the beginning of the buffer as proposed in [18]. This way, the buffer itself can be fully used for allocating the payload. We call the gadgets that allow to move the stack *Stack_move*.

Given that the amount of data injected is limited, exploiting the same vulnerability multiple times could place the attacker in an advantage position. However, exploiting a buffer overflow usually leaves the stack in a unpredictable state and the attacker is usually forced to reset the device each time to maintain the device functional and/or resume its normal operation. In this regard, Francillon and Castellucia [17] proposed to perform a software reset by directly jumping to the address $0x0000$ (i.e. the reset vector). However, this approach is not suitable in modern AVR chips since it does not guarantee that the the I/O registers are restored to their initial state [19]. Therefore, we propose the use of a gadget, namely *Reset_chip*, that uses a *Watchdog reset* (which is one of the reset sources used in AVR chips). More precisely, the gadget first establishes a watchdog timer and then jumps to an infinitive loop. When the timer expires, the watchdog provokes a software reset.

Figure 3 depicts the schematic view of a generic data injection attack. When the vulnerable function is called, the return address is pushed on the stack. Thus, the attack starts by overwriting this address with the address of the gadget *Stack_move* (Step 1), which pops the new address and stores it in the memory address corresponding to the stack pointer (SP). From there on, the buffer constitutes the new stack (Step 2). Then, the address of the next gadget is popped from the stack, so the first bytes of the buffer must point to the gadget *Store_data* (Step 3) that stores the data at a given address (Step 4). As showed in Section III-B, both the stored data and the SRAM memory address must be included in the payload. Finally, when the gadget *Store_data* returns (Step 5), the program jumps to the gadget *Reset_chip* (Step 6), which performs a clean software reset of the AVR chip. The adversary, while needed, may send new payload to exploit the vulnerability and store additional data in consecutive addresses. In every reboot, the *.data* and *.bss* sections of the SRAM memory are cleared and reloaded, so if the adversary stores the data in a memory area different from these (i.e. the region tagged as *unused* in Figure 2), then data remain persistent across reboots.

Finally, once the required data are stored in memory, the adversary is ready to call the library function using a chain of gadgets that perform the desired operation. First, she must load the arguments and prepare the data required (e.g. pointers to objects) using the data injection scheme explained above. Second, the program flow must jump to the desired function itself.

B. Exploit implementation in Arduino Yun

In this section, we describe the implementation of the exploit targeting Arduino Yun devices, that allows an adversary to execute remote commands in the OpenWrt environment (i.e.: bypassing the *Bridge* between the two chip-sets). The attack comprises two phases: *injection*, and *invocation*. First, it starts by injecting the command into SRAM memory as a String object, and then forces the execution of the function *runShellCommand(String* cmd)* from the *Bridge* library[4] by passing as arguments a pointer to the injected object.

[4]https://www.arduino.cc/en/Reference/YunProcessConstructor

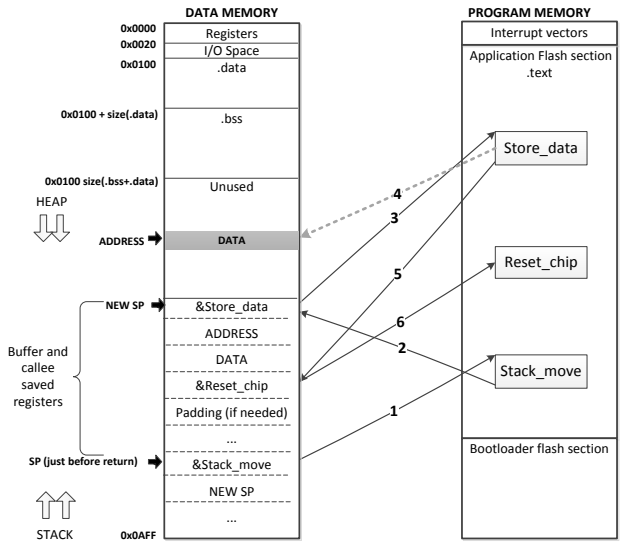

Fig. 3. Scheme of the data injection ROP attack

In this work, we have exploited a function (implemented ad-hoc for the prototype) that receives data from the Bluetooth port and stores it into a buffer, without checking its bounds. By sending a crafted data, we are able to overwrite the return address of the function and take control of the program flow. Next, we explain the different settings and implementation details of the attack.

We use a pair of gadgets used to move the Stack Pointer (SP) to a given address[5]. The first gadget (*pop r29; pop r28,; ret*) loads the new SP to registers r28 and r29 and the second gadget (*out 0x3e, r29; out 0x3d, r28*) stores the SP in 0x3e and 0x3f, which are actually the positions mapping the SP. This is possible because AVR uses fixed positions of data memory to store I/O registers, including the SP. Gadgets used to move the stack are very frequent in AVR binaries, since they are used to save and restore the stack within the called functions.

To store the data in SRAM, we have found an optimal couple of gadgets, showed in Table I. These gadgets are included with the String library, imported by default in all Arduino programs, show it is reasonable that the adversary can use it *at will*. As these gadgets are consecutive in the code, they can be used recursively. In the first interaction, the gadget *Load_data* at address 0x2c00 loads data in registers r16 and r17, and the destination address in registers r28 an r29. In AVR, registers r28 and r29 are mapped to the register Y used for direct addressing. Here, the gadget *Store_data* showed in Table I uses the fixed displacement of the Y register to store the values from r16 and r17 in addresses Y+2 and Y+3 respectively. Because the end of the gadget *Store_data* directly jumps to the gadget *Load_data*, they can be used repetitively, as shown in Figure 4.

To perform a software reset of the AVR chip we use one of the reset sources provided by the AVR architecture, the Watchdog reset, which establishes a timeout and resets the chip when it expires. Concretely, a first gadget enables the

[5]We only show the relevant instructions from the gadgets

Address	Instructions	Description
	Store_data	
0x2bf6	std Y+3, r17 std Y+2, r16 ldi r24, 0x01 rjmp .+2	Stores the values from r17 and r18 in addresses Y+3 and Y+4 (Y is the concatenation of the registers r29 and r28). Then, jumps to 0x2c00.
	Load_data	
0x2c00	pop r29 pop r28 pop r17 pop r16 ret	Loads the new values at r17 and r16 and new addresses at r28 and r29

TABLE I
GADGETS USED TO STORE DATA IN A GIVEN ADDRESS OF THE SRAM

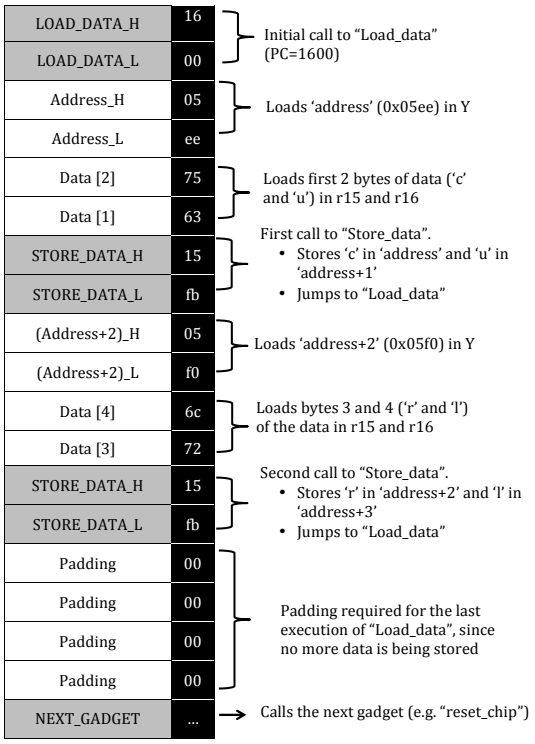

Fig. 4. Schematic view of a payload that inserts the command "curl" (0x63,0x75,0x72,0x6c) into a the address 0xef00 of SRAM memory using the gadgets from Table I.

Watchdog (using the instruction *wdr*), and sets a timeout to 120ms. Then, a second gadget performs an infinitive loop, and is intended to wait the timer to expire (this gadget, which consists on just one instruction, is the last instruction of every Arduino program, and represents the "stop-program" instruction to maintain the device in an *idle* state). By chaining these two gadgets, the chip automatically resets an the normal operation of the Arduino device is restored. Then, the adversary may send a new exploit to store more data, depending on what she wants to inject.

Finally, once the adversary stores the command to be executed in the SRAM (e.g. "curl", as shown in Figure 3), the exploit calls the function *runShellCommand* of the *Bridge* Library. This function takes as argument the address of the String object which represents the command, which is provided in registers. We use a gadget (consisting in a set of *pop* instructions) to perform such loading. Then, the program flow should directly jump to the *runShellCommand* function

which uses the *Bridge* between the two chips to execute the desired command in the OpenWrt. As explained before, these commands are executed with full privileges since they are issued by the root user.

IV. Design and behavior of ArduWorm

The exploitation described in Section III shows that IoT devices with limited resources, such as those based on AVR architecture, can be a weak point in a IoT infrastructure. This presents a big threat to both organizations and end users, as devices such as Arduino Yun can be compromised and used as entry vector to attack other devices in the same network. Moreover, in the case of ArduWorm, the vulnerability is specially dangerous as the attacker is able to execute arbitrary commands in the target machines with *root* privileges.

Based on this initial exploitation, we built a worm namely ArduWorm. In the next sections, we first describe the behavior and structure of the worm, and then we provide two attack scenarios where this worm could be specially harmful.

A. ArduWorm

ArduWorm takes advantage of the position IoT devices usually have within a network in order to spread and infect devices which are local to the network. ArduWorm is coded in Python, and it includes a Remote Access Tool (RAT) for controlling infected machines. As any other worm, ArduWorm is composed of four main stages, namely payload execution, persistence, reconnaissance and propagation. ArduWorm forks into two processes at startup. One of them executes the RAT and listens for commands, whereas the other performs the persistence, reconnaissance and propagation stages.

1) Payload: The RAT payload of ArduWorm opens port 16333 and provides a shell to the attacker. These commands are being executed with *root* privileges, so the attacker can install third-party elements in the machine such as network spoofers or sniffers.

Additionally, since the payload is being executed with *root* privileges, it leaks the */etc/shadow* file containing user names and password hashes to an external, more powerful server. The external server uses some password cracking program (such as the popular John the Ripper [20]) in an attempt to get the clear passwords from the hashes. Whenever those passwords are cracked, the server sends them back to the RAT for further usage during the propagation phase.

2) Persistence: During the persistence stage, ArduWorm looks for information regarding the machine where it is being executed. It detects the type of machine and devices connected to it and its current level of privileges. If ArduWorm has not enough privileges to persist, it will stop this phase and move on to the next one.

If enough privileges are granted, ArduWorm copies itself to the system library and creates a startup script in */etc/init.d*. In this way, the worm will be executed on each reboot. It will also create a new user in the system for a remote attacker to log in via SSH. This allows remote access even in the case where the RAT backdoor (i.e., the port 16333) is filtered by a perimeter firewall.

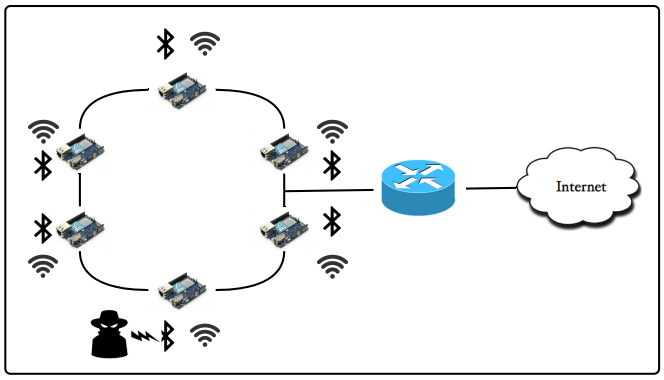

Fig. 5. WSN attack scenario

3) Reconnaissance: After persisting, ArduWorm gets all active network interfaces and performs a network scan to locally reachable devices. ArduWorm will look for TCP ports 22 and 23, usually bound to SSH and Telnet services. ArduWorm tries to connect to such ports, and if a response is received, it stores the IP/port pair for its usage during the propagation phase. ArduWorm also performs TCP/IP stack fingerprinting [21] in order to infer the operating system running such machines. Depending on the information gathered, ArduWorm will attempt to propagate to such machines. Currently, the worm capabilities are only targeted to other Linux-based devices.

ArduWorm also checks for serial ports connected to the device interfaces, such as bluetooth dongles. These serial interfaces are different from the ones controlled by the Atmega32u4 MCU, and can be used during the propagation phase as explained in the next section.

4) Propagation: Once potential targets have been identified, ArduWorm starts the exploitation stage if the hashes leaked during the RAT setup have been already cracked and returned from the server. Given the huge amount of sensors deployed in a typical IoT scenario, it can be assumed that devices may share the same or similar passwords. Thus, by knowing the password from the current machine, it may be possible to gain access to neighbor devices by performing a small dictionary attack (e.g. by appending a number to the cracked password). If successful, ArduWorm will copy and execute itself in the next device, repeating the whole process again.

Additionally, ArduWorm includes a script to send the same exploitation technique described in Section III. For such a purpose, it would be trying to compromise serial interfaces connected to the Atmega32u4 system on chip, which may be directly connected to other devices based on the AVR architecture, for example using a serial bluetooth. Indeed, if the Arduino devices were connected forming a sensor network, there is a chance of another sensor similar to the one being controlled by Arduino to be connected in such serial port, making a further exploitation stage possible.

B. Attack PoC

To show the threat exposed by ArduWorm, we propose two proof-of-concept scenarios where it could be spread.

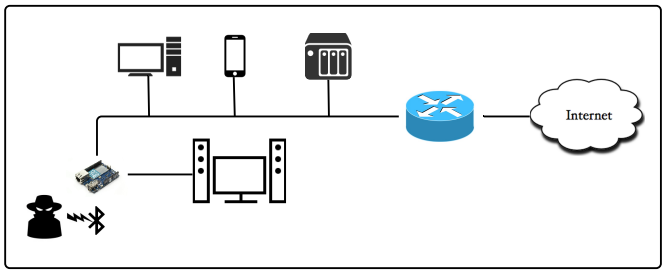

Fig. 6. Home network attack scenario

First, suppose that a Wireless Sensor Network (WSN) used for meteorological forecast is deployed in the field to measure different environmental-related data over time. Such sensors are inter-connected between them and to the Internet in order for their operator to be able to receive and configure the sensors remotely. Each sensor is controlled by an Arduino Yun, being the Atmega32u4 chip in charge of reading data from the sensor itself, and the OpenWrt being responsible of the controlling of such sensor parameters and information forwarding through the network. Additionally, the sensors have a serial bluetooth interface connected to the Atmega32u4 chip in order for operators to configure the sensor when no network connectivity is available (e.g. to configure the network during the initial deployment). Figure 5 depicts the previously described scenario.

By using ArduWorm, an attacker may be able to compromise the bluetooth serial interface connected to the Atmel chip in one of the sensors. It can then run arbitrary commands in the OpenWrt side of the Arduino Yun controller with *root* privileges. This can be used to issue a command to download the ArduWorm Python executable and run it on the machine. Once ArduWorm is present in one device inside the network, it will spread itself to other sensors as explained above.

In addition to the previously proposed scenario, consider a second situation in which a user has installed an Arduino Yun at home in order to control devices attached to it (see Figure 6). In this case, the Arduino Yun board is used in order to act as a gateway between the network attached storage (NAS) device and the audio music system, so it is possible for him to act play songs stored in the former machine.

Again, if an attacker is able to gain access to a serial interface in the Arduino Yun Atmega32u4, then she may be able to compromise the Linux system of the board. By downloading ArduWorm, she could use the RAT capabilities to control the OpenWrt system, and to install tools which allows her to control other devices in the network. This compromised Arduino Yun can thus spoof all traffic in the network or impersonate the home router, making all communications in the home network of the user vulnerable to MITM attacks.

V. Possible countermeasures

Since the IoT market is projected to grow notably in the next few years, it is expected that exploits and malware targeting IoT devices and networks grow as well. Thus, it is essential to be prepared and provide further security checks to the developed applications and firmwares. In this section we provide an overview of countermeasures that could be devised against the attack proposed in this work.

A. Protection against memory corruption vulnerabilities

Currently, micro-controllers based on AVR architecture lack of mechanisms to protect against well known memory corruption attacks, such as buffer overflows and control flow hijacking. Unfortunately, AVR devices mainly consist of tiny devices with limited processing power. Moreover, they rely on a monolithic processor that can only run one process at a time, and that do no accept runtime modifications. This makes modern defenses such ASLR or Stack-canary protection inappropriate.

In order to avoid code reuse attacks such as those presented in this work, research community should focus on lightweight mechanisms which modifies the memory layout of the flash memory. For example, Habibi et al. [18] presented a method that periodically randomized the memory layout of a AVR MCU to avoid ROP attacks, by using external hardware support.

B. Network protection

IoT devices are intended to be connected either locally (e.g. LAN or Wifi networks) and globally to the Internet. Thus, it is essential to provide these devices to proper network protection mechanisms, such as encryption or packet authentication. While this seems obvious, lack of encryption and authentication mechanisms are two of the top 10 vulnerabilities found by OWASP for IoT [22]. It would be desirable a standardized security protocol for the success of IoT. When every object in our daily life is connected to the Internet, they must speak the same (security) protocol to ensure interoperability. The standardization efforts have to grow with the technology, giving rise to a very important effort to make IoT a reality.

C. Overcoming the gap of diversity

The IoT paradigm requires the connection of a widespread diversity of devices, having different architectures and hardware equipment. For example, this difference is obvious in the wearable devices connected to a smartphone using bluetooth. A smartphone can afford almost any bluetooth authentication scheme, but if the device which is being paired lacks of display and/or keyboard, then the pairing must be done with an unsecure ”just-works” mechanism [23] which is vulnerable to Man-in-the-Middle (MITM) attacks.

In IoT scenarios it is possible for an adversary to compromise the weakest point to further spread to other devices. Indeed, during our experimentation using the Arduino Yun we have found a difference between the capabilities of the Atmega32u4 (AVR) chip and the Atheros (Openwrt) one. Accordingly, by exploiting a vulnerability in the Atmega32u4 we have been able to control the Openwrt, which in turn can lead to a more severe attack. It is indispensable that extra security measures are taken in the OpenWrt. For example, by authenticating the commands sent through the bridge connecting both chips, or by limiting the commands that can be triggered from the Atmega32u4 chip (in its current state, the commands are issued by the *root* user, which has full privileges on the system). It would be desirable if the Arduino community could provide a OpenWrt release containing such countermeasures to be flashed on future Arduino Yun devices.

VI. Conclusions

The Internet of Things is an emerging scenario that is widely used in critical ITC scenarios such as medical monitoring, automotive and smart homes. Thus, the consequences of security breaches extend from classical loss of privacy, confidentiality and integrity of digital assets, toward physical harm to human beings. It is essential to equip IoT devices and networks with proper security mechanisms to hinder malicious activities.

In this work we have presented a functional malware specimen (i.e. ArduWorm) which is able to compromise Arduino Yun devices, which are common in the IoT arena due to its low cost and ease of use. By exploiting a vulnerability in the resource constrained AVR chip integrated in Arduino Yun, we showed how an adversary can hijack the OpenWrt chip that has full connectivity capabilities. This is possible due to the flaws encountered in the design of the *Bridge* library that communicates both chips and that do not provide access control neither authentication. We have described a proof of concept worm which installs a backdoor and provides the adversary with a Remote Access Tool (RAT) to the device.

We hope that this work can help the Arduino community to modify the *Bridge* library to hinder similar attacks. Moreover, we consider that our work motivates research and development of lightweight security mechanism that can be implemented in resource constrained devices such as Atmel AVR chips.

Acknowledgments

This work was 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).

References

[1] Pa, Y. M. P., Suzuki, S., Yoshioka, K., Matsumoto, T., Kasama, T., Rossow, C.: "IoTPOT: analysing the rise of IoT compromises". In *9th USENIX Workshop on Offensive Technologies (WOOT)*, 2015.

[2] Proofpoint: "Proofpoint Uncovers Internet of Things (IoT) Cyberattack". [online] Available at: *http://investors.proofpoint.com/releasedetail.cfm?releaseid=819799* [Accessed March. 2016].

[3] Wojtczuk, R.: "The advanced return-into-lib (c) exploits: Pax case study". In *Phrack Magazine*, Volume 0x0b, Issue 0x3a, Phile# 0x04 of 0x0e, 2001.

[4] Roemer, R., Buchanan, E., Shacham, H., Savage, S.: "Return-oriented programming: Systems, languages, and applications", In *ACM Transactions on Information and System Security (TISSEC)* 15(1), 2, 2012.

[5] Snow, K.Z., Monrose, F., Davi, L., Dmitrienko, A., Liebchen, C., Sadeghi, A.R.: "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization". In *IEEE Symposium on Security and Privacy 2013*, pp. 574–588, 2013.

[6] McAfee corporation: "McAfee Labs Threats Report: August 2015". [online] Available at: *http://www.mcafee.com/us/resources/reports/rp-quarterly-threats-aug-2015.pdf* [Accessed March. 2016].

[7] Panda: "PandaLabs Annual Report 2015". [online] Available at: http://www.pandasecurity.com/mediacenter/src/uploads/2014/07/Pandalabs-2015-anual-EN.pdf [Accessed March 2016].

[8] Symantec corp.: "The dawn of ransomwear: How ransomware could move to wearable devices". [online] Available at: http://www.symantec.com/connect/blogs/dawn-ransomwear-how-ransomware-could-move-wearable-devices [Accessed February 2016]

[9] Hernandez, G., Arias, O., Buentello, D., Jin, Y.: "Smart nest thermostat: A smart spy in your home", in *Black Hat USA*, 2014.

[10] Bill Hau, Tony Lee, Josh Homan.: "SynFul Knock - A Cisco Router Implant". In *Fire Eye Threat Research, Advanced Malware*, 2015.

[11] Ballano, Mario: "Is there an Internet-of-Things vigilante out there?". [online] Available at: *http://www.symantec.com/connect/blogs/there-internet-things-vigilante-out-there* [Accessed March 2016].

[12] Halperin, Daniel, Thomas S. Heydt-Benjamin, Benjamin Ransford, Shane S. Clark, Benessa Defend, Will Morgan, Kevin Fu, Tadayoshi Kohno, and William H. Maisel.: "Pacemakers and implantable cardiac defibrillators: Software radio attacks and zero-power defenses." In *IEEE Symposium on Security and Privacy*, pp. 129-142, 2008.

[13] Camara, C., Peris-Lopez, P., and Tapiador, J. E.: "Security and privacy issues in implantable medical devices: A comprehensive survey". In *Journal of biomedical informatics*, 55, 272-289, 2015.

[14] Miller, Charlie, Valasek, Chris.: "Remote exploitation of an unaltered passenger vehicle". In *Black Hat USA*, 2015.

[15] Checkoway, S., McCoy, D., Kantor, B., Anderson, D., Shacham, H., Savage, S., Koscher, K., Czeskeis, A., Roesner, F., and Kohno, T.: "Comprehensive Experimental Analyses of Automotive Attack Surfaces". In *USENIX Security Symposium*, 2011.

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

[17] Francillon, A., Castelluccia, C.: "Code injection attacks on Harvard-architecture devices". In *Proceedings of the 15th ACM conference on Computer and Communications Security* pp. 15–26. 2008.

[18] Habibi, J., Gupta, A., Carlsony, S., Panicker, A., Bertino, E.: "MAVR: Code reuse stealthy attacks and mitigation on unmanned aerial vehicles". In *IEEE 35th International Conference on Distributed Computing Systems (ICDCS)*, pp. 642–652, 2015.

[19] Atmel corp.: "AVR Libc Reference Manual". [online] Available at *http://www.atmel.com/webdoc/AVRLibcReferenceManual* [Accessed November 2015].

[20] Alexander Peslyak. "John the Ripper password cracker". [online] Available at: *http://www.openwall.com/john/* [Accessed March 2016].

[21] Erik Hjelmvik. "Passive OS Fingerprinting". [online] http://www.netresec.com/?page=Blog&month=2011-11&post=Passive-OS-Fingerprinting [Accessed 11 Mar. 2016]

[22] Miessler D. and Smith, C.: "Top 10 IoT Vulnerabilities (2014) Project". In *The OWASP Foundation*, 2014.

[23] Haataja, K., and Toivanen, P.: "Two practical man-in-the-middle attacks on bluetooth secure simple pairing and countermeasures", In *IEEE Transactions on Wireless Communications*, 9(1), 384-392, 2010.