Computing and Informatics, Vol. 43, 2024, 529–560, doi: 10.31577/cai_2024_3_529

FORENSIC ANALYSIS OF THE IOT OPERATING SYSTEM UBUNTU CORE

Juan Manuel Castelo Gómez

University of Castilla-La Mancha
Avda. de España s/n
02071, Albacete, Spain
e-mail: juanmanuel.castelo@uclm.es

José Roldán-Gómez

University of Oviedo
Federico García Lorca 18
33007, Gijón, Spain
e-mail: roldangjose@uniovi.es

Sergio Ruiz-Villafranca, Álvaro del Amo Mínguez

University of Castilla-La Mancha
Avda. de España s/n
02071, Albacete, Spain
e-mail: sergio.rvillafranca@uclm.es, alvarodel.amo@alu.uclm.es

Abstract. The number of cyber incidents in which the Internet of Things (IoT) device or system is present is increasing every day, requiring the opening of forensic investigations that can shed light on what has occurred. In order to be able to provide investigators with proper solutions for performing complete and efficient examinations in this new environment, IoT systems and devices are being studied from a forensic perspective so that tools and procedures can be designed accordingly. In this article, besides reviewing the proposals from the community on this matter, the multi-purpose IoT operating system Ubuntu Core is studied to determine in what way a forensic investigation of this system should be performed, detailing how

to approach the acquisition and analysis phases. In addition, both the volatile and non-volatile artifacts that might hold useful information are listed and described, and a forensic tool is presented for their recovery as well as for the acquisition of the non-volatile memory.

Keywords: Internet of Things, IoT forensics, digital forensics, cybersecurity

Mathematics Subject Classification 2010: 68-M99

1 INTRODUCTION

The emergence of the Internet of Things (IoT) as a new environment in which to conduct forensic investigations has introduced a great variety of new systems and devices that had never been analyzed before. Computers and smartphones have given way to smart switches, televisions, cars, and personal assistants. New contexts, such as eHealth, smart cities, and smart industries, have appeared, something that was unimaginable a few years ago. The technology which once was reserved for certain scenarios has now been transformed and implemented in our everyday life, being present in almost every aspect of it.

As a result, forensic investigators find it extraordinarily difficult to conduct an investigation in this environment. Although all these contexts belong to the IoT, they are quite dissimilar to each other and have been developed to perform very different tasks. Aspects such as the operating system or firmware they run, whether their memory is soldered onto the board or the way in which an investigator can access them, are crucial for properly performing an examination. For example, the approach that needs to be followed when analyzing a device running a real-time operating system (RTOS) which has a soldered storage is not the same as when studying a general-purpose operating system (GPOS) with a removable memory. Therefore, in order to provide investigators with guidelines on how to deal with them, the research community studies IoT devices and describes what information can be recovered from them and how to do so.

In addition, these studies allow the community to develop solutions for conducting forensic investigations, such as methodologies or tools that, due to the characteristics of the IoT, need to be adapted to the new environment. Unfortunately, the number of IoT-centered solutions is still low and that hinders the examination process. But this also works the other way round, as if there is not enough information on how IoT devices work, which data they handle, how to recover them or how they interact with each other, that means the solutions cannot be developed properly. Furthermore, the progression of such solutions will set the standards allowed in court if an IoT investigation is involved in a legal process.

In view of this, the forensic analysis of new IoT operating systems and devices, especially the most widely used ones, can be useful for acquiring knowledge

of the behaviour of this type of devices, and can shed some light on how to approach the development of solutions, ultimately improving IoT forensics as a whole. Furthermore, such analysis is also useful for modelling the context in which these devices are used, and for finding similarities and differences with others. Therefore, the community not only benefits by having guidelines on how to examine a system, but also by being provided with future areas and viewpoints on which to work.

In this regard, this paper presents a forensic analysis of the multipurpose IoT operating system Ubuntu Core, which is used in IoT gateways, dome cameras, smart mirrors, bench top sequencers and single-board computers, among others. In addition, it is based on one of the most widely used Linux distributions for desktops and servers, so the authors believe that its examination is of interest for the forensic community.

Contributions. The main contributions of this study are as follows:

- We present a review of the proposals from the community regarding the forensic analysis of IoT devices and systems.
- We perform the analysis of a forensically speaking unexplored operating system, namely Ubuntu Core, studying its static and dynamic behavior.
- We detail how to carry out the acquisition and analysis phases, addressing both the offline and remote methods for each one when handling the three main types of evidence: non-volatile memory, volatile memory and network traffic.
- We explain how the data are distributed in Ubuntu Core, and we list the relevant information that can be retrieved from the operating system and which may be useful in a real investigation. This serves as a guideline to quickly observe which data can be extracted from the operating system, how to do it and where they are located.
- We develop a tool to collect the relevant artifacts found in the non-volatile and the volatile memory. In addition, it allows the acquisition of the raw data stored in the non-volatile memory.

The rest of the paper is organized as follows. A brief description of the Ubuntu Core operating system is presented in Section 2. Section 3 discusses the proposals from the research community regarding the forensic analysis of IoT devices and systems, together with its challenges. The methodology followed to carry out the research is described in Section 4. Section 5 details how to carry out the acquisition and analysis of the IoT operating system, and Section 6 lists the forensic relevant information found after performing said investigation. A tool for the recovery of these forensic artifacts and the raw data in non-volatile memory is presented in Section 7. Finally, our conclusions are presented in Section 8.

2 UBUNTU CORE

Ubuntu Core is the IoT operating system developed by Canonical, and it was released for the first in 2014. It is based on the Ubuntu desktop and server versions, two of the most widely used Linux-based distributions in their respective categories [1], although it has many fundamental concept differences, as seen in Table 1. It can be run in several IoT platforms, with Table 2 showing those with certified compatibility and has been designed to be a flexible operating system that can be used in multiple contexts, such as vehicle infotainment [2], but primarily in two: industrial settings and smart homes. This is evidenced by its use on IoT gateways, dome cameras, smart mirrors or bench top sequencers, among others. The main features of this operating system are the following:

- It allows the execution of a version of Ubuntu on resource-constrained devices built from Snap packages.
- It uses the snap daemon (snapd) to govern the system's configuration, package management, and update control. Unlike Advanced Package Tool (APT), snaps are self-contained packages that run in sandbox mode, thus do not communicate directly with the host.
- It is compatible with Bluetooth connectivity.
- It supports the creation of custom system images.
- It provides access to several IoT applications such as servers, home, machine to machine (M2M) gateways, and radio access network platforms.
- It uses snap applications to provide functionality to the device. They can be programmed in C, C++, Python, Java, Node.js and Go.
- It can be configured automatically using model assertions.
- It provides an app store named Snap Store, from which multiple tools and servers can be installed.
- It allows remote access to the system via Secure SHell (SSH) by using a public key linked with an Ubuntu Single Sign On (SSO) account, which is downloaded when the system is set up.
- It supports access to a real-time-kernel.
- It provides bare metal cloud support through Metal-As-A-Service (MAAS).
- It is compatible with the installation of a graphical interface, although it has none by default [3].

3 RELATED WORK

Before focusing on the forensic studies which address the IoT context, it is essential to understand how the operating systems and the distribution from which Ubuntu

Feature	Ubuntu Core	Ubuntu Desktop
Application execution	Applications and software are self-contained and isolated in secure sandboxes	Less strict confinement measures for applications that rely on system-wide dependencies
Application development	Specific frameworks and software development kits that integrate IoT protocols and standards	Wide range of programming languages and frameworks
Updates	Atomic updates that can be rolled back and are applied automatically	Applied individually, without rollback capability
User interface	Headless	Full-featured graphical interface
Authorization	User based on Ubuntu SSO account with full permissions	Freedom to manage user and set permissions as desired
System management	Using command-line through SSH	Both command line and desktop environment
Marketplace for software and applications	Snap Store	Ubuntu Software Center

Table 1. Comparison between Ubuntu Core and Ubuntu Desktop

Core originates work. Since their appearance, the Linux and UNIX-based operating systems have been a topic of interest for forensic investigators. This has led to carrying out extensive studies on what data is handled by them, such as [4] and [5] providing insights on how the file system, memory, logging system or booting process operate, and discerning which forensic artifacts can be extracted, as well as which commands allow their retrieval. Likewise, the Ubuntu distribution has undergone similar analysis, with works such as [6] examining its file system, listing the most relevant directories and files stored in it, and presenting an evidence collection tool that can extract the user's activity or generate a timeline.

Focusing on the IoT, the study of its devices and systems has proven to be quite useful to determine how to approach forensic investigations. This has resulted in guidelines that assist on examinations in a heterogeneous environment with different characteristics to what investigators were used to. A proposal which addresses the study of an IoT operating system from a forensic perspective is [7]. In it, a common methodology for conducting investigations on IoT prototyping hardware platforms is proposed and tested on the Raspbian [8] operating system, listing the directories and file locations that provide essential information sources for the investigator. A command-line tool is also introduced that allows the acquisition of these data and generates a *.csv* file which stores the hash value of each artifact, their modified access and creation times, and their size.

A similar study is performed in [9], in which the Windows 10 IoT Core operating system is forensically analyzed, and the relevant information to be found in it is

Model	Supported Ubuntu Core Versions
Raspberry Pi Models 2, 3, CM 3 and CM 3+	Ubuntu Core 22, 20, 18 and 16
Raspberry Pi Model 4, CM 4 and Lite	Ubuntu Core 22, 20, and 18
Intel NUC	Ubuntu Core 22, 20 and 18
Raspberry Pi 400	Ubuntu Core 22 and 20
Raspberry Pi Zero 2 W	Ubuntu Core 22
ASUSTek PE100A	Ubuntu Core 20
Advantech UNO-127, UNO-2271G, UNO-420	Ubuntu Core 20
Avnet AVTSE-RPI-IIOTG	Ubuntu Core 20
Element Biosciences Instrument Aviti System	Ubuntu Core 20
Honeywell HC70WB8R2, HC70W48R2 and HC70WZ5130	Ubuntu Core 20
Lenovo ThinkEdge SE30 and SE50	Ubuntu Core 20
Qualcomm DragonBoard	Ubuntu Core 18 and 16
Intel TANK-870-Q170	Ubuntu Core 18
FORME Life Studio	Ubuntu Core 18
Dell Edge Gateway 3001, 3002, 3003 and 5000	Ubuntu Core 16
Rigado Cascade 500	Ubuntu Core 16

Table 2. Platforms with certified compatibility with Ubuntu Core

listed. In addition, a module is developed for the KAPE [10] tool, which allows the extraction of data marked as relevant during the analysis, using an image or a clone of the non-volatile memory as a source.

Focusing on another IoT operating system, [11] studies the file system used by Contiki OS. Apart from analyzing its structure and dynamic behaviour, the authors present a tool to reconstruct file versions from a memory dump that operates in two stages. Firstly, it extracts files and their fragments from all memory pages, categorizes them, and then detects other versions of that files by measuring the similarity between them. This tool is tested in a simulated environment using the Cooja simulator and achieving good results.

Finally, in [12] the forensic study targets Linux-Compatible platforms, specifically Tizen and Linux. Similarly to the approach followed in the previous research, the authors examine several characteristics of the file systems used, such as the metadata, type, size, files, and folders in it. In addition, they perform an experiment to determine the possibility of checking file names by file system, extracting them, and recovering deleted files.

With the aim of showing the differences between these already forensically-studied IoT operating systems and Ubuntu Core, the one under the examination in this article is presented Table 3.

One of the most challenging aspects of IoT investigations have proven to be the acquisition phase. With storage now being soldered to the board, accessing the data handled by these devices has become more difficult. In addition, having the physical access to the devices is not a guarantee, and not all systems provide means

Feature	Ubuntu Core	Contiki OS	Windows 10 IoT Core	Raspbian	Tizen OS
Platform	ARM, x86, and others	Tiny low-power microcontrollers	ARM devices	Raspberry Pi	ARM, x86, and MIPS
App Management	Snap	Cooja simulator	Windows Update	APT	Tizen Package Manager
User Interface	Headless	GUI	GUI	GUI	GUI
Network Protocols	Wi-Fi, CoAP, AMQP, Zigbee, Z-Wave, and Bluetooth	LR-WPAN, 6LoWPAN, CoAP and RPL.	Wi-Fi, NFC, LoRa, and Bluetooth.	Wi-Fi and Bluetooth	Wi-Fi, Bluetooth, and NFC
User	User management linked with Ubuntu SSO	No user management	Built-in user management	Built-in user management	Built-in user management

Table 3. Comparison between Ubuntu Core with other IoT operating systems forensically-studied by the research community

of remotely connecting to them. Therefore, new approaches are needed to extract data.

In [13], this issue is addressed focusing on consumer and industrial IoT devices. Although the purpose of the research is to demonstrate that these devices are vulnerable to certain attacks, the information provided is useful from a forensic perspective. This is due to the acquisition method that the authors use for their two case studies in which they examine two devices from each context, corresponding to the Joint Test Action Group (JTAG) and the Universal Asynchronous Receiver/Transmitter (UART). These methods are fairly common in smartphone forensics, but have been replaced by the use of hardware tools, which do not exist in the IoT, therefore it is important to know their feasibility in this environment. By using the methods, the authors are able to dump the EEPROM memory and modify some parameters of the devices to attack them.

Another interesting acquisition method mostly used on smartphones is tested on an IoT device in [14], namely the chip-off. In this proposal, a forensic analysis of the new TomTom navigation devices is carried out, describing the techniques which allow dumping the memory and detailing how to decode its data. One of these methods is chip-off, which consists in desoldering the memory chip and placing it in a reader. Chip-off is not one of the most recurrent options for investigators due to its

complexity and risk, in addition it requires specific equipment and soldering knowledge to be able to perform. The other method presented, which is only compatible with specific versions of certain TomTom devices, consists in wiring certain points of the memory chip to an SD card reader. Once the data is accessible, information such as the last GPS position, the home location or the Bluetooth device connected can be retrieved.

Chip-off and JTAG are also feasible methods for acquiring data from smart vehicles, as described in [15]. Besides, detailing the challenges associated with vehicle forensics and listing some generic and specific tools that can be used for their examination, two case studies are presented – one performing a forensic analysis of the entertainment system of a Volkswagen Golf car, and the other studying the mobile traffic data from several vehicles. In the first experiment, after determining the type of system present in the car by scanning it using on-board diagnostics (OBD), the multimedia device is extracted, confirming that the JTAG and chip-off are compatible. Information such as the chassis number or engine control unit (ECU) serial number can be recovered. With respect to the second experiment, after capturing the mobile traffic data, information such as the location, chassis number and car status can be accessed.

With the JTAG technique as a base, [16] presents a memory acquisition framework for Industrial Control Systems (ICS). Using the hardware information of the device, the proposal is able to create a JTAG profile that can be used for acquiring and analyzing the contents of the non-volatile memory. The framework is tested on an Allen-Bradley 1756-A10 device hosting a 1756-L61 controller, successfully acquiring a dump of the memory together with its profile.

Also focusing on Programmable Logic Controllers (PLC) acquisition, but opting for a completely different approach and technique, [17] introduces a framework, PEM, for remotely collecting their volatile memory. Using the PLC's communication protocol, the control logic of the device is infected with a memory duplicator that copies the memory contest to a free space in it, which can be accessed over the network. In addition, the process is performed while the PLC is normally operating. The effectiveness of the proposal is tested by using it for collecting a Schneider Electric M221 PCL in a scenario simulating an attack.

As we can see, not only operating systems are the source of study, some works focus on addressing specific IoT contexts. This approach can gather the knowledge on how the most frequently used IoT systems work – which can be extrapolated and used in other ones.

One example is [18] that reviews the forensic procedures available for the examination of Small Scale Digital Devices (SSDD). It addresses smartphones, drones, gaming console, wearables and smart toys. For all of them, the authors mention the acquisition methods available, the artifacts that can be found in them, and the challenges such devices present for investigators, among other information.

Concentrating on the smart home context, [19] presents a forensic model that is tested on the Amazon Echo smart assistant. With this analysis, the authors are able to analyze the data type created, transmitted, processed and store by the device and

others that interact with it, being able to examine the network traffic exchanged, the firmware of the Amazon Echo, and the app used to control it.

In this case, also addressing IoT consumer devices, but focusing on fitness trackers, [20], describes how to acquire and analyze the data from two devices, namely the Xiaomi Mi Band 2 and Fitbit Alta HR. In addition, the significant data which can be extracted from each one is listed, and guidelines are provided on how to examine deleted and modified data from such devices.

Finally, in [21], the retrieval and analysis of the log of an intelligent robot vacuum system and the app used to control it is described. Information such as the app installation date and time, its version number, the usage events, the clean schedule of the robot, the user credentials and network information can be extracted from both devices.

After reviewing a range of proposals from the community regarding the forensic analysis of multiple IoT contexts and their devices, the following conclusions can be drawn:

- There is a worrisome lack of IoT-centered tools, which hinders the investigation process. Therefore, until they are developed, investigators must rely on conventional tools to perform examinations.
- Regarding the acquisition process, methods such as JTAG, UART or chip-off have become more feasible since the storage is usually soldered to the device's board, added to the fact that there are not any hardware solutions that can be used to assist in this task. However, these techniques cannot always be carried out, and they require specific equipment and knowledge, especially in the case of chip-off, which also has a high chance of compromising the functioning of the device.
- The interaction with the IoT device means that several other devices apart from the IoT device might need to be examined as well. The cloud is the most usual site to appear in this scenario, but investigators can also find smartphones or computers. Consequently, it might be useful to study the data that are exchanged between them, which usually can only be performed through a live or remote analysis, as information is exchanged on-the-fly without being stored.
- The forensic analysis of the data extracted from IoT devices shows that conventional tools allow investigators to obtain sufficient information to be able to carry out investigations. In addition, the data that can be extracted from each context and their form are quite dissimilar, which means that the approach might need to change depending on the context in which the investigation is taking place.

4 METHODOLOGY

To understand how the operating system distributes the data, three different scenarios were studied, with each one representing a different state of the operating

system. By doing this, we are able to approach the analysis gradually, avoiding to miss the study of possible useful pieces of data, rather than facing a scenario in which all the data are examined at once. See Figure 1. The three different scenarios:

- The image file is written to the storage. It allows us to study the provisioning files and the general structure of the storage before the system boots for the first time. With respect to the latter, this also allows doing so without having to determine which data were generated by the user and which are specifically used by the system to work.
- The system boots for the first time. Through this analysis, we evaluate the system once the initial configuration finished. In this scenario, we encounter the first data that were generated by the user, namely the network configuration and the Ubuntu SSO account linking that is required for the user to connect to the device and for the system to work. In addition, we study how the data change from the first scenario with respect to this one, as well as we do so with the main services and process executed by the system, taking the advantage of the fact that there are none purposely launched by the user.
- The system is used in a normal scenario. Lastly, the goal is to study the data generated by the operating system when the user interacts with it. To achieve this, all the features of the system are explored, some of them being the following: establishing a connection between an external computer and the Ubuntu Core system, applications and snaps are installed, deployed, restored and deleted, snapshots are created, and external devices are paired.

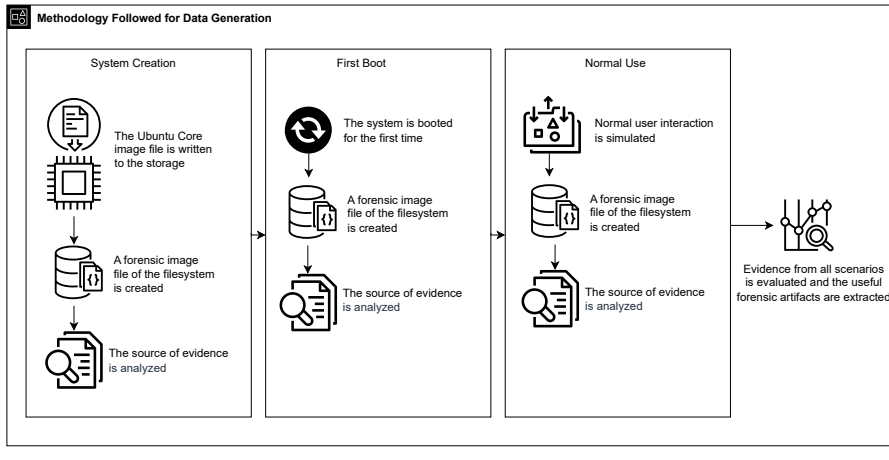


Figure 1. Methodology followed for performing the forensic analysis

Test Environment. In order to carry out the analysis, it is necessary to establish and configure a proper environment to make sure that the experiment is performed correctly. In our case, the components used are the following:

- Raspberry Pi Model 3 B+ [22]: host of the Ubuntu Core operating system.
- 32 Gigabyte microSD Card: non-volatile memory of the IoT devices used, as they do not include a soldered storage unit.
- Ubuntu Core 22, 20, 18 and 16: all the available releases of the long-term support (LTS) version of the IoT operating system Ubuntu Core [23]. Each one is individually tested following the methodology described above. Depending on the version, the compatibility with the different platforms varies, so Table 2 which releases are supported by all of them.
- External PC: operates as the forensic computer, which has all the necessary tools installed on it. It also contains the public key needed to connect to the Ubuntu Core operating system via SSH.
- Operative Wi-Fi and wired network: needed to study the effects of using a network on the device. We used a router executing the OpenWRT [24] operating system in order to make sure that we could easily capture network traffic from it if needed.

5 FORENSIC ANALYSIS OF UBUNTU CORE

In this section, a description of how to approach the acquisition and analysis phases of the investigation is presented, describing both the physical and remote methods for the three main types of evidence that can be studied, namely storage, volatile memory and network traffic.

5.1 Acquisition

In order to carry out the data acquisition process when investigating the Ubuntu Core operating system, there are two techniques that can be carried out: a physical acquisition, and a remote acquisition. The physical acquisition refers to the process of collecting the data stored in the non-volatile memory used by the platform in which the Ubuntu Core operating system is running by physically interacting with its storage while it is off. On the other hand, a remote acquisition consists of executing the acquisition software directly on the device by connecting to it using a remote service, thus allowing access to the non-volatile memory, the volatile one, and the network traffic. In this experiment, all the available forensic techniques for each have been tested, describing below the procedure and their feasibility.

5.1.1 Physical Acquisition

Carrying out a physical acquisition depends on the IoT platform in which the data is stored. As described in Section 2, there are many platforms that are compatible

with Ubuntu Core, so, with the goal of summarizing which acquisition technique can be executed in each one is presented in Table 4. The specific procedures that can be carried out are the following:

Extraction and acquisition. Only feasible if the storage is removable. This is the simplest method of acquisition, and the most common in conventional forensics. The storage device, usually a microSD card or a drive, is extracted from the system, placed in a write blocker to preserve its integrity, and then either cloned or imaged.

JTAG/UART. A method that involves connecting to the Test Access Ports (TAPs) of the memory using a JTAG connector in order to be able to read its data and image it. It is normally a harmless option for soldered storage, and can also be used on non-soldered ones, but the compatibility of the device with the JTAG is not guaranteed.

In-System Programming (ISP). This involves connecting to an eMMC or an embedded Multi Chip Package (eMCP) flash memory chip to access its content. It is quite similar to the JTAG method, also requiring a connector, and the method is usually non-destructive as well.

Chip-off. The memory is desoldered from the board and placed into a flash reader, and then its image file is created. It requires specific soldering knowledge and equipment. Furthermore, the chances of compromising the functioning of the device are quite high.

5.1.2 Remote Acquisition

The main disadvantage of performing a remote acquisition is that the interaction with the system will alter the data stored on it, and there are no guarantees that the collection tool will be compatible with it. However, if the integrity does not have to be preserved, it might be preferable to performing a JTAG or chip-off, as it is faster, simpler, and it does not damage the device. On the other hand, it is the only option available if the device cannot be physically accessed or if the physical acquisition methods cannot be carried out.

Furthermore, with this technique the investigator might be able to access all three main types of evidence: storage, volatile memory and network traffic. The feasibility of the process does not depend on the platform being used, but on the operating system, so the details provided in this section apply to all the IoT devices compatible with Ubuntu Core. In this proposal, the authors tackled the collection of said types of evidence, achieving the following results:

Non-volatile memory. The approach to performing a remote acquisition of the storage is identical to any other Linux distribution, as the *dd* command is included by default in Ubuntu Core. Therefore, once the Universally Unique IDentifier (UUID) of the storage has been determined, which can be done using either the mount or *blkid* commands, the acquisition can be performed by

Model	Storage Device	Extraction & Acquisition	JTAG	Chip-off
Raspberry Pi Models 1, 2, 3, 4	MicroSD card	✓	Not recommended	Not recommended
Raspberry Pi Model CM 3	Soldered eMMC flash memory	✗	✓	✓
Intel NUC	M.2 or a 2.5-inch drive	✓	✗	✗
ASUSTek PE100A	MicroSD card and soldered eMMC flash memory	✓	✓	✓
Advantech UNO-127 and UNO-2271G	mSATA drive and soldered eMMC flash memory	✓	✓	✓
Advantech UNO-420	MicroSD card and soldered eMMC flash memory	✓	✓	✓
Avnet AVTSE-RPI-IIOTG	Soldered eMMC flash memory	✗	✓	✓
Element Biosciences Instrument AVITI System	Soldered eMMC flash memory	✗	✓	✓
Honeywell HC70WB8R2, HC70W48R2 and HC70WZ5130	Soldered eMMC flash memory	✗	✓	✓
Lenovo ThinkEdge SE30 and SE50	M.2 or SSD drive	✓	✗	✗
Qualcomm DragonBoard	MicroSD card and soldered eMMC flash memory	✓	✓	✓
Intel TANK-870-Q170	2.5-inch drive	✓	✗	✗
FORME Life Studio	Soldered eMMC flash memory	✗	✓	✓
Dell Edge Gateway 3001, 3002 and 3003	MicroSD card and soldered eMMC	✓	✓	✓
Dell Edge Gateway 5000	M.2 drive	✓	✗	✗
Rigado Cascade 500	Soldered eMMC flash memory	✗	✓	✓

Table 4. Summary of the feasibility of each physical acquisition method for each compatible platform

executing dd. In addition, apart from having the option of saving the resulting image file in an external USB storage, *netcat* is also available by default, so it can also be sent to a third device connected to the same network.

Volatile memory. Several conventional memory acquisition tools such as *LiME* [25], *lmg* [26] or *fmem* [27] were tested in this experiment, but they were not able to read the kernel module of the system that is needed for the application which creates its own module that ultimately allows access to the memory. The authors tried to install the kernel headers and the required packages manually, but the operation was unsuccessful. Furthermore, in order to compile the tools, it was necessary to install the classic environment on the system, as there is no native compiler included in it, and a snap-based one is not available for installation, which generates a great amount of data that have no forensic value and altering the system. Even directly accessing the memory using dd was tested, as it used to work on older versions of Linux systems, but with no success. As a last resort, the authors tried to make a cross-compilation on an emulated ARM machine which had the same kernel as Ubuntu Core, but the tools could not be compiled on this system either. However, this experiment was useful to demonstrate how the lack of proper tools can compromise an examination, and to show that not all IoT systems and devices can be studied using conventional forensic solutions.

Network traffic. If the investigator opts to capture the data directly from Ubuntu Core, *tcpdump* [28] is one of the tools that can perform this operation, but first the classic environment needs to be installed on the system in the form of a snap in order for it to work. After that, the application can be installed as in any other Ubuntu distribution using apt-get. However, it is also possible to capture the traffic from a different device in the network, normally a router or a switch. The procedure is quite similar, but it does not require interacting with the IoT device, thus preserving its integrity. Its downside is that it will also acquire traffic from the other devices in the network. There is also a hardware-based approach, consisting in using a router or switch with port mirroring capabilities and then capturing the packets sent through the interface to which the IoT device is connected.

5.2 Analysis

With respect to the study of the system generated data, some guidelines are provided on how to approach each analysis method, mentioning which tools and commands can be used when examining Ubuntu Core. Firstly, the offline approach is addressed, which uses a previously acquired forensic file or clone as the source data. Secondly, the remote method is detailed. The former is performed by remotely connecting to the system using SSH while the latter is done by debugging, which can only be done in the platforms which allow that. In this case, all are listed in Table 2 except from the Element Biosciences Instrument Aviti System, Honeywell

HC70WB8R2, HC70W48R2 and HC70WZ5130, FORME Life Studio, and Rigado Cascade 500.

5.2.1 Offline Analysis

Once the storage has been either imaged or cloned, the investigator can treat the data source like any other traditional one. To analyze it, since there are no IoT-centered tools, conventional ones can be used to browse through the directories and, in the case of the latter, to carve the deleted files from the storage. Although the investigator cannot take the advantage of all the functionalities offered by these tools, as they are centered on handling data from conventional sources, they provide enough information to perform the analysis.

Regarding the network traffic, the resulting capture file can be analyzed with conventional tools as well.

5.2.2 Remote Analysis

This method provides the opportunity to study all three types of data. Taking advantage of the native SSH service and that most platforms are debugging compatible, it is quite easy for the investigator to carry out this type of examination, which is not something usual in IoT devices. The result of this method studying each source of evidence is as follows:

Non-volatile memory. A remote analysis is a less interesting method than an offline one in terms of the information that can be extracted from the non-volatile memory. The data available to study is the same in both techniques, but when offline some external software can be used to assist in the process, so it is faster and easier to browse through the contents of the file system. On the other hand, there are few native commands that provide summarized information regarding certain aspects of the system, especially snaps, so it speeds up certain processes of the analysis.

Volatile memory. Dynamic information regarding the processes in the system, virtual file systems, runtime and temporal data can be accessed by following a remote approach. In order to have access to more tools, it is recommended to install the classic environment. However, as previously discussed, this installation has a bad impact for the system's integrity, as the number of packages installed and data generated is quite high. Therefore, performing this action is only recommended when it is not necessary to preserve the integrity of the evidence in the investigation, or if there is no other analysis method available.

An alternative method is debugging. This requires connecting to specific ports in the chip using an adaptor, which allows accessing the data by connecting it to a computer that executes the debugging tool. Unfortunately, extracting valuable information using this method is highly unlikely, as it requires analyzing data at instruction level, so the authors would recommend investigators to use the

native commands instead, which are more likely to present useful data for the investigation.

Network traffic. There are not many native tools available to study this type of evidence, so, in order to perform an analysis in real time of the network traffic, the classic environment needs to be installed and use external tools such as *iptraf* [29], *iftop* [30] or *netperf* [31]. The downside of this approach is that normally only general information about the traffic that is being exchanged by the device is provided by these tools. While it is quite useful to get a general idea on how the network works, to get packet-level data *tcpdump* must be used, which shows every packet but in a very user-unfriendly manner. Under these circumstances, it is better to opt for an offline analysis if the investigator wants to study the network traffic at a packet-level.

As a summary of the static and dynamic information that can be extracted from Ubuntu Core, Table 5 presents the most useful native commands which can be used in a remote analysis.

Command	Description
mount	Lists all mounted file systems
ps -r	Shows all running processes ordered by user
df	Displays the space available on the file systems and their mount point
dmesg	Prints the boot up messages, which are not displayed on the screen
snap list –all	Shows the list of all installed snaps, also displaying their revisions
snap changes	Shows the recent system changes regarding the snaps
snap info <snap_name>	Shows additional details of a snap, such as its description, ID, or the version installed
snap connections <snap_name>	Shows the interfaces being used by a snap
snap services	Shows information about the services in all the installed snaps or a specific one
snap logs <snap_service>	Shows the logs from a snap's services
lastlog	Provides information on when the users last logged into the system

Table 5. Useful native commands for performing an online/remote analysis of Ubuntu Core

6 FORENSIC INFORMATION AND ARTIFACTS IN UBUNTU CORE

In this section, the most relevant information which was detected during the analysis of the operating system that would be interesting from a forensic perspective is listed and summarized, describing their purpose, content, and location.

6.1 System Structure

It is crucial to know how the data are distributed in the storage, which is detailed in Table 6. As any common Linux distribution, when running, Ubuntu Core combines both virtual and physical file systems. Apart from the traditional some new ones can be found. The first one is "/meta", which contains the metadata information for the base snap package of the Ubuntu release. The second one is "/snap", which is a symbolic link to */writable/system-data/snap*, and contains files and folders from installed snap packages. Finally, there is a virtual file system named "/host", whose purpose is unknown, as there are no data stored in it, and only appears in the latest version of the operating system. With the aim of providing a detailed guideline on the contents of the operating system, Table 7, explains all the file systems in it and their purpose.

6.2 Physical Storage

The physical file system for data storage is located in the system in the */writable* directory. Consequently, when performing an offline analysis, it would be the source of evidence to be examined. It is distributed in the following way:

- A directory for the system data named *system-data*, which contains the root home folder, and the snap configuration and their data folder as well as for other services and programs. In addition, it is also the location in which the logs are stored.
- A directory for the user data named *user-data*, which contains the home folder for the Ubuntu SSO account user and its personal configuration for the snaps and services.

6.3 Process and Services

Although it is not possible to acquire the volatile memory and then perform an extensive offline analysis of it using forensic memory tools, a bit of information can be extracted using the native commands provided by the system to determine how Ubuntu Core behaves dynamically, which can be useful to detect an anomaly in the investigation. The usual behaviour of the operating system is described below.

- The only processes launched at user level are the *systemd* instance that manage the user services, a child process for the Pluggable Authentication Modules (PAM), which allow the user to log in to the system, and the SSH service.
- The rest of them are launched at the super-user level and are used to start services such as the wireless connection, the snap package manager and *systemd*, which then starts the processes associated with journaling, network configuration, time synchronization, kernel, domain resolution and user login manager.

Ubuntu Core 22 & 20			
Partition	File System	Size	Description
ubuntu-seed	FAT32	1.2 GB	It contains the overlays needed for the hardware to work, as well as the base snaps of the system. It is mounted in */var/lib/snapd/seed* when the system boots as a read-only file system
ubuntu-boot	FAT32	750 MB	Partition used for the system to boot. It stores the kernel for the Raspberry Pi, its image, the drivers to be loaded in RAM to boot, and a file with data regarding the model of the operating system. It is mounted in */run/mnt/ubuntu-boot* when the system boots
NONAME	ext4	16 MB	It contains the assertions which describe the policies for the device. It is mounted in */var/lib/snapd/save* when the system boots
NONAME	ext4	Remaining space	It stores the system and user data. It is mounted in */writable* when the system boots
Ubuntu Core 18 & 16			
Partition	File System	Size	Description
ubuntu-boot	FAT32	256 MB	It contains the same data as in the newest version, except for the description file, plus the overlays
NONAME	ext4	Remaining space	It has the same data and purpose as the last partition of the Ubuntu Core 20 version

Table 6. Partitions into which the storage is divided

In addition, as usual in any Linux operating system, the */proc* directory offers some relevant information regarding each individual process that is running in the system. Some of the data which can be extracted from this directory is as follows:

- Information about the file systems that are mounted in the system can be found in the file */proc/mounts*. Additionally, data with respect to the blocks of each partition both virtual and physical is presented in */proc/partitions*, and more concrete information for each file system, such as the number of inodes being used or the options assigned to it, is located in */proc/fs/*.
- Workload for the memory, CPU and IO devices is saved in */proc/pressure*.

Directory	Description
bin	It is a symbolic link to */usr/bin*, containing the binaries executables by any user
boot	It stores the files needed for the system to boot
dev	It contains the files which represent the different devices in the system
etc	In it, configuration files for services and programs are stored
home	It is the route in which the personal directory of the user is located
host	It is a directory which only appears in this version of the distribution, and only in the latest release, but no data have been stored in it, therefore its purpose is unknown by the authors
lib	It a symbolic link to */usr/lib*, and contains the shared libraries needed for the system to work
media	It is the location in which the removable media is usually mounted
meta	A location only used by this operating system which contains the metadata information for the base snap package of the Ubuntu release, namely "core20"
mnt	It is the directory in which temporary file systems are mounted
opt	It is the location in which software packages are normally installed in Linux, but it is not used in Ubuntu Core
proc	It handles processes and system data
root	It is the personal directory for the superuser
run	It stores temporal data in runtime
sbin	It is a symbolic link to */usr/bin*, and stores system binaries, as well as the ones only executables by a superuser
snap	It is a symbolic link to */writable/system-data/snap*. It contains files and folders from installed snap packages
srv	It usually stores data for services provided by a Linux system, but it is empty in Ubuntu Core
sys	It contains information regarding system components such as drivers and kernel features
tmp	Temporary files used by programs are located in this directory
usr	It is a read-only file system which stores all user utilities, such as libraries, binaries, and documentation
var	It contains writable system files which are modified during runtime, such as logs
writable	It is the location in which the physical storage is mounted

Table 7. Structure of the root file system

- In */proc/devices* data regarding the devices which are connected to the system is stored.
- Files containing network stats, such as the Address Resolution Protocol (ARP) entries, Wi-Fi connection, packets sent and received by each network adapter and socket in use are located in the */proc/net* directory.
- Data regarding the state and configuration of the General-Purpose Input/Output (GPIO) pins is stored in */proc/device-tree/__symbols__*, finding another directory for the override pins in */proc/device-tree/__overrides__*.
- For each process, information regarding the command that launched it, the environment variables that it is using, the file systems that it has mounted, its state, and a symbolic link to its current working directory can be found in */proc/<PID>* in the files *cmdline*, *environ*, *mounts*, *stat*, and *cwd*, respectively.

6.4 Network

The file which contains the network configuration is located in */writable/system-data/etc/netplan/00-snapd-config.yaml*. An interesting aspect of the file is that it shows the password of the access point to which it is connected. As mentioned in Section 4, both a wired and a wireless connection were configured. Another relevant artifact is */writable/system-data/etc/hosts*, the local file for domain translations, which is sometimes used by malware to connect to remote domains. By simple interacting with the system no other network information can be obtained using native commands, but if the classic environment is installed, typical Linux commands such as *ifconfig* or *netstat* can be executed.

In addition, when the system is running, that information is stored in */run/systemd/network*, while data about the DHCP service can be found in */run/systemd/leases*, which informs of the lease assigned to the device.

6.5 Users

The only user who can access the system is the one that was linked with the Ubuntu SSO account during the first boot configuration. The name of the user matches the username's account and has the ability to execute commands with super-user privileges through *su*, since the root account is not protected. No other users can be either created or deleted, although more do exist, but those are the system users who are created by some applications or by the operating system in order for them to work properly, which can be found in the usual */etc/passwd* file, as well as the groups, in */etc/group*.

As mentioned, the home directory of the user is located in the ext4 partition, specifically in the */writable/user-data directory*, which is mounted in the system when it boots in */home*, while the root directory is located in */writable/system-data/root/*. Apart from the files stored by the user, the bash history can be found

in this route, as well as the file which contains the public keys associated with the Ubuntu account which are authorized to log into the system, which are stored in the personal configuration directory for the SSH service, namely *.ssh/authorized_keys*. This has additional relevance since manually adding a public key to this file allows the device which is holding that key to connect to the Ubuntu Core system even though if that key is not associated with the user's Ubuntu SSO account.

In addition, the data regarding the Ubuntu SSO account can be found in *.snap/auth.json*, showing its username and the associated email account.

6.6 SSH

SSH is one of the most relevant services provided by Ubuntu Core, as it is the only way for both to interact and remotely connect to the system by default. In order to do so, the user must connect with the username of their Ubuntu SSO account and use its associated public key associated as an authentication method. Several keys can be associated with one account, but only that individual account can be used to connect.

The route in which the SSH files are stored is */writable/system-data/etc/ssh/*. Among others, both the server and the client configuration file and the public and private key files of the host can be found here. Unless the logging level is changed from info, which is the default mode, to verbose, not much more data can be retrieved. However, by using the *btmp*, *lastlog* and *tallylog* system log files, which are located in */writable/system-data/var/log*, information about failed login attempts, the last login and the number of failed login attempts can be extracted. Additionally, the last login information for each user in the system can also be extracted by using the *lastlogin* command.

If the investigator is performing a remote analysis, information regarding the SSH session which is currently opened can be found in the */run/systemd/sessions*.

6.7 Snaps

This is the largest source of information about the system, and the one which varies the most with respect to the desktop and server versions of Ubuntu, since snap is the package manager by default in the IoT one. The snaps installed can be found in */writable/system-data/snap/bin*. The data generated by the applications are stored in the user's home directory, specifically in the route */writable/user-data/<username>/snap*. In addition, the cached data of the installed snap are stored in */writable/system-data/var/lib/snapd/cache*, and the snapshots, which can be created automatically by default, are located in */writable/system-data/var/lib/snapd/snapshots/*.

Using the native command *snap* also provides a great amount of data when performing a remote analysis. Some of the useful information that can be extracted

like the list of installed snaps and their services, all the changes undergone by them, the interfaces that they are using or specific logs for each snap.

6.8 Logs

Apart from the logs from the snaps, there are not many other logs that can be found in the system, as there are not many services running. However, a few ones can be found in */var/log/*, which are described below:

- A log with information regarding the active session in the system, which has the name *1*. The IP address from which the connection has been established can be found, as well as the user and service that have created it, which would be the Ubuntu SSO account and the *sshd* service, among other data.
- The messages printed by the system when it booted for the first time can be found in a file named *install-mode.log*.
- Data regarding subiquity, which is the first boot installer, are stored in *console-conf*.
- As mentioned above, the *btmp*, *wtmp* and *lastlog* files are also located in this directory providing information about failed login attempts, the last login and the number of failed login attempts.

As a summary, Table 8 is presented, in which a list of the most relevant artifacts detected in the physical storage of Ubuntu Core and their description can be found. In addition, in Table 9 the volatile ones which can be studied when performing a remote analysis are listed as well.

7 TOOL FOR THE COLLECTION OF RELEVANT FORENSIC ARTIFACTS

With the aim of addressing the lack of solutions to perform forensic analysis in IoT devices and systems, this section proposes a tool for the retrieval of the relevant artifacts that are present in the non-volatile and volatile memory of any device running the Ubuntu Core operating system, which have been described in Section 6, as well as for the acquisition of the raw data stored in the non-volatile memory, more specifically in the *writable* partition. Its description, together with the explanation of its usage and an example of its execution, is presented below.

7.1 Tool Description

Although it is possible to execute only one of the tasks described above, with the aim of providing the highest degree of detail, this explanation details the process of executing all of them. This does not significantly change the execution flow as the tool comprises modules that work independently. It is up to investigators to

Evidence Description	Location
User's home directory	/user-data/username
Root's home directory	/system-data/root/
Bash history	/user-data/username/.bash_history
User's Ubuntu Single Sing On account information	/user-data/username/.snap/auth.json
Authorized keys associated with the Ubuntu Single Sign On account which are allowed to log into the system via SSH	/user-data/username/.ssh/authorized_keys
Network configuration	/system-data/etc/netplan/00-snapd-config.yaml
Local file for domain translation	/system-data/etc/hosts
SSH server configuration	/system-data/etc/ssh/sshd_config
Public key used by the SSH host	/system-data/etc/ssh/ssh_host_rsa_key.pub
Snapshots of snaps created	/system-data/var/lib/snapd/snapshots
Cache of the installed snaps	/system-data/var/lib/snapd/cache
Local data from the snaps	/user-data/<username>/snap
Last logged session's IP	/system-data/var/log/lastlog
Number of failed logins	/system-data/var/log/tallylog
Information regarding failed logins	/system-data/var/log/btmp
Log of the subiquity service	/system-data/var/log/console-conf
Messages printed by the system when it first booted	/system-data/var/log/install-mode.log

Table 8. Most relevant artifacts found in the physical storage of Ubuntu Core

choose which task suits their investigation better. The following actions are carried out:

- The tool checks if the arguments are correct and if there is connectivity with the target device. If everything is correct, it shows the user's menu in which a task to perform can be selected.
- Once an option is selected, it tries to establish a connection through SSH. If the connection is successful, the operations for performing the artifacts' collection and/or the filesystem acquisition are performed. Otherwise, the execution halts.
- If the filesystem acquisition is to be performed, a directory with the name "Image File" is created in the device that launched the tool which will contain the resulting image file with the raw data of the *writable* partition. In addition, a log file informing of the process's result as well as of the characteristics of the acquired filesystem is created. It is in the .txt form and contains the following information:
  - Size of the partition.
  - Date and time in which the acquisition started.

Evidence Description	Location
Configuration of the GPIO pins	/proc/device-tree/__symbols__
Override GPIO pins	/proc/device-tree/__overrides__
File systems mounted in the system	/proc/mounts
Virtual and physical partitions in the system	/proc/partitions
Details of each file system in the system	/proc/fs/
Command which launched a process	/proc/<PID>/cmdline
Environment variables being used by a process	/proc/<PID>/environ
File systems used by a process	/proc/<PID>/mounts
Data regarding the state of a process	/proc/<PID>/stat
Current working directory for a process	/proc/<PID>/cwd
Workload for the memory, CPU and IO devices	/proc/pressure
Data regarding the devices connected to the system	/proc/devices
Network configuration	/run/systemd/network
Network stats, such as packets, ARP entries and Wi-Fi connection	/proc/net
DHCP leases	/run/systemd/leases
Information regarding the open SSH session	/run/systemd/sessions
User information	/run/systemd/users

Table 9. Most relevant volatile artifacts found in Ubuntu Core

  - Date and time in which the acquisition finished.
  - MD5 and SHA1 hash codes for the resulting image file.
  - Name of the image file generated.

  Both files have the date and time in which the acquisition started as a filename. Additionally, in order to transfer the data from the target device to the one that launched the tool, the output of the *dd* command is sent directly to the latter.

- To perform the collection of the non-volatile artifacts, the tool connects to the target device through the SSH File Transfer Protocol (SFTP) and performs a logical copy of the directories and files of interest.
- Finally, if the volatile artifacts are to be collected, firstly the tool executes the commands that perform a remote analysis of the device, saving their output to an individual .txt file per command. Secondly, as it is done with the non-volatile artifact collection, a SFTP connection is established and the files are copied logically to the device that launched the tool. In this case, since there are a high number of them, they are organized in directories depending on the information that they present. In order to keep all artifacts in a structured way, two directories are created, one for the command outputs, and one for the files collected.
- Once the option selected by the investigator is completed, the tool shows the menu again (or exits) if all three tasks have been performed.

The tool[1] uses Python as a base language, and it is divided into six modules that operate in the following way:

- *tool.py.* Main module checks whether the arguments are correct, presents the selection menu to the user, and launches the corresponding modules.
- *ssh_connection.py.* It performs the connection to the device using SSH.
- *filesystem_collection.py.* It executes the corresponding commands to acquire the *writable* partition, and generates the log file.
- *non_volatile_acquisition.py.* Collects the non-volatile artifacts listed in Table 8.
- *volatile_acquisition.py.* Executes the commands listed in Table 5, and collects the volatile artifacts detailed in Table 9.
- *file_collection.py.* Executes the commands that collect the relevant files in the system and stores them in the device which launched the tool, organizing them in different directories depending on their type.

Evidence Preservation. In order to ensure that the execution of the tool is performed in a way in which the integrity of the data stored in the system is best protected, the measures described below are taken.

- If the user wants to execute the three different tasks available so that all artifacts, together with *writable* filesystem, are collected, the processes are launched in the order from the less intrusive to the most intrusive one in terms of modifying the data stored in the system. As a result, first the acquisition of the filesystem is performed, then the collection of the non-volatile artifacts, and finally the collection of the volatile artifacts, which requires executing commands in the system, thus altering its original state.
- Each time the tool is launched, it is checked whether the device which launches it has already stored pieces of evidence in that directory, halting the process if that is the case. This way, the tool offers protection against accidental re-executions that may overwrite previously collected pieces of evidence.
- Any task carried out by the tool, except for the launch of the commands that extract information regarding the dynamic behaviour of the operating system, do not generate any data in the target system, it only does so in the device in which the tool was launched.
- Finally, a log is generated for the acquisition of the *writable* filesystem, which allows the investigator to check the integrity of the evidence.

[1] The tool's source code is publicly available in the following repository: `https://bitbucket.org/juanmanuelcastelo/ubuntu-core-forensic-artifact-collection-tool/src/master/`

7.2 Usage and Execution Example

The tool is launched by executing the *tool.py* module, for which the following arguments are required:

- IP address of the Ubuntu Core device, which is specified with the argument *"-a"*.
- Private key file of the Ubuntu SSH account, which is specified with the argument *"-k"*.
- Username, which is specified with the argument *"-u"*.

In Figure 2, an example of an execution is shown in which the three types of evidence are collected, and how they are stored in the device which launches the tool.

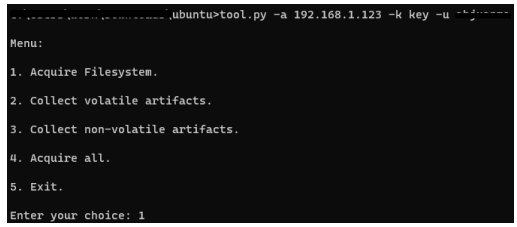
8 CONCLUSIONS

In this paper, we have addressed IoT forensics and how the research community is dealing with the emergence of the IoT and the systems and devices that comprise it in order to develop solutions for conducting complete and efficient forensic investigations. In this regard, one of the approaches followed is the study of the systems and devices with the purpose of understanding what information they contain and how to extract it, provides the investigators with guidelines how to examine the information.

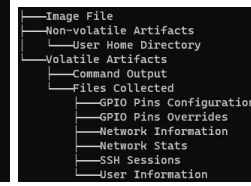
After reviewing the proposals from the community regarding the analysis of IoT devices from different contexts, we have identified the techniques that could be applied for the examination of systems in general, as well as it was understood what approach researchers follow when they perform these studies.

As a result, the IoT operating system developed by Canonical, namely Ubuntu Core, was selected as an interesting one to examine, since it can be used in many IoT contexts and has a multipurpose nature. In addition, it is based on one of the most widely used operating systems in the desktop and server environment, a fact that may ultimately lead to Ubuntu Core being one of the most widely used systems in the IoT.

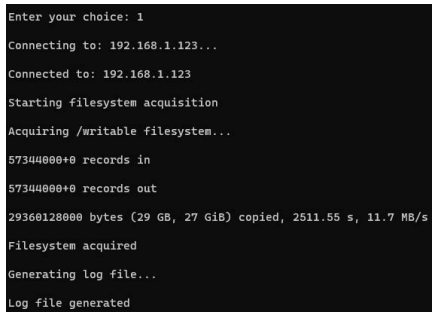
We had looked at the dynamic and static behavior of Ubuntu Core and identified how the system distributes its data. Furthermore, the process that can be conducted for acquiring and analyzing this operating system has been detailed, describing how to approach both the remote and offline methods. During this analysis, it became clear that the lack of IoT forensic tools, and in particular those compatible with Ubuntu Core, severely hindered the volatile memory examination process, in fact we were not able to acquire it at all. In addition, this issue also affected the usefulness of performing a remote analysis of the system, since the investigator must rely on its native tools, which do not provide much information.



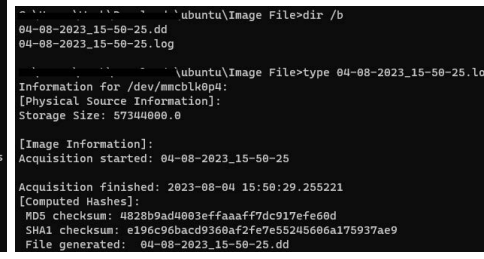
a) Tool execution example and its options



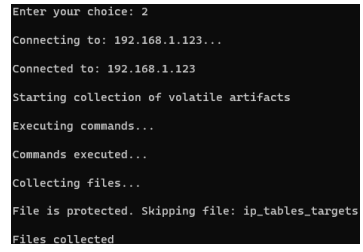
b) Tool's output with the structure of all the pieces of evidence collected



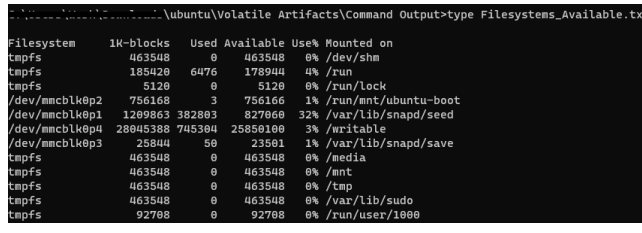
c) Execution of the filesytem's acquisition task



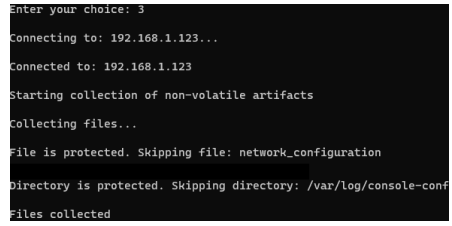
d) Image file and log generated after the acquisition is completed



e) Execution of the volatile artifacts' acquisition task



f) Example of the output of a launched command



g) Execution of the non-volatile artifacts' acquisition task

h) Example of a collected file containing the authorized keys to sign in the system

Figure 2. Example of the execution of the tool for the collection of the forensic artifacts and its output

After carrying out this analysis, the useful forensic artifacts found have been listed and described, detailing their location and how to access the information they present. That can serve as a handbook to be used by investigators in future examinations of Ubuntu Core. Finally, a tool has been developed for the retrieval of these artifacts as well as the acquisition of the raw data stored in the non-volatile memory.

8.1 Future Work

As mentioned above, there is a wide spectrum of research regarding IoT forensics that requires attention. Some projects that could be addressed are the following:

- Gather the knowledge extracted from this forensic analysis, apply the knowledge gained from studying similar systems belonging to the same context, the aim to design the methodology for conducting investigations.
- Develop additional solutions compatible with Ubuntu Core and similar IoT operating systems to improve the effectiveness of the analysis and facilitate this task for investigators, especially for acquiring and analyzing the volatile memory.
- Perform further forensic analysis of IoT systems so that the community has more information on how to deal with them, and what approach to follow in the solutions development in order to help investigators.

Acknowledgements

This research was supported by the University of Castilla-La Mancha under the contract No. 2022-PRED-20677, and the project No. 2022-GRIN-34056, by the Spanish Ministry of Economic Affairs and Digital Transformation under the project No. PID2021-123627OB-C52, and by the Regional Government of Castilla-La Mancha under the project No. SBPLY/21/180501/000195.

REFERENCES

[1] W3Techs: Usage Statistics and Market Share of Linux for Websites, March 2023. 2023, https://w3techs.com/technologies/details/os-linux.

[2] Ubuntu: Pelagicore Signs Up to Ubuntu Core for in-Vehicle Infotainment Development. 2011, https://bit.ly/3wFL2IT.

[3] Get Started with Ubuntu Core. 2024, https://ubuntu.com/core/docs/get-started.

[4] Altheide, C.—Casey, E.: Chapter 6 – UNIX Forensic Analysis. In: Casey, E., Altheide, C., Daywalt, C., de Donno, A. et al. (Eds.): Handbook of Digital Forensics and Investigation. Academic Press, 2010, pp. 301–351, doi: 10.1016/B978-0-12-374267-4.00006-9.

[5] Ling, T.: The Study of Computer Forensics on Linux. 2013 International Conference on Computational and Information Sciences, 2013, pp. 294–297, doi: 10.1109/ICCIS.2013.85.

[6] Patil, D. N.—Meshram, B. B.: Digital Forensic Analysis of Ubuntu File System. International Journal of Cyber-Security and Digital Forensics, Vol. 5, 2016, No. 4, 175–186 pp., doi: 10.17781/p002213.

[7] Bharadwaj, N. K.—Singh, U.: Acquisition and Analysis of Forensic Artifacts from Raspberry Pi an Internet of Things Prototype Platform. In: Sa, P. K., Bakshi, S., Hatzilygeroudis, I. K., Sahoo, M. N. (Eds.): Recent Findings in Intelligent Computing Techniques (ICACNI 2017). Springer, Singapore, Advances in Intelligent Systems and Computing, Vol. 707, 2019, pp. 311–322, doi: 10.1007/978-981-10-8639-7_32.

[8] Raspberry Pi OS. 2022, https://www.raspberrypi.org/downloads/raspberry-pi-os/.

[9] Castelo Gómez, J. M.—Roldán Gómez, J.—Carrillo Mondéjar, J.—Martínez Martínez, J. L.: Non-Volatile Memory Forensic Analysis in Windows 10 IoT Core. Entropy, Vol. 21, 2019, No. 12, Art. No. 1141, doi: 10.3390/e21121141.

[10] Zimmerman, E.: Introducing KAPE - Kroll Artifact Parser and Extractor. 2019, https://www.kroll.com/en/insights/publications/cyber/kroll-artifact-parser-extractor-kape.

[11] Sandvik, J. P.—Franke, K.—Abie, H.—Årnes, A.: Coffee Forensics – Reconstructing Data in IoT Devices Running Contiki OS. Forensic Science International: Digital Investigation, Vol. 37, 2021, Art. No. 301188, doi: 10.1016/j.fsidi.2021.301188.

[12] Lee, J.—Shon, T.: Forensic Analysis of IoT File Systems for Linux-Compatible Platforms. Electronics, Vol. 11, 2022, No. 19, Art. No. 3219, doi: 10.3390/electronics11193219.

[13] Wurm, J.—Hoang, K.—Arias, O.—Sadeghi, A. R.—Jin, Y.: Security Analysis on Consumer and Industrial IoT Devices. 2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC), 2016, pp. 519–524, doi: 10.1109/ASPDAC.2016.7428064.

[14] Elstner, J.—Roeloffs, M.: Forensic Analysis of Newer TomTom Devices. Digital Investigation, Vol. 16, 2016, pp. 29–37, doi: 10.1016/j.diin.2016.01.016.

[15] Le-Khac, N. A.—Jacobs, D.—Nijhoff, J.—Bertens, K.—Choo, K. K. R.: Smart Vehicle Forensics: Challenges and Case Study. Future Generation Computer Systems, Vol. 109, 2020, pp. 500–510, doi: 10.1016/j.future.2018.05.081.

[16] Rais, M. H.—Awad, R. A.—Lopez Jr, J.—Ahmed, I.: JTAG-Based PLC Memory Acquisition Framework for Industrial Control Systems. Forensic Science International: Digital Investigation, Vol. 37, 2021, Art. No. 301196, doi: 10.1016/j.fsidi.2021.301196.

[17] Zubair, N.—Ayub, A.—Yoo, H.—Ahmed, I.: PEM: Remote Forensic Acquisition of PLC Memory in Industrial Control Systems. Forensic Science International: Digital Investigation, Vol. 40, 2022, Art. No. 301336, doi: 10.1016/j.fsidi.2022.301336.

[18] Hosani, H. A.—Yousef, M.—Shouq, S. A.—Iqbal, F.: State of the Art in Digital Forensics for Small Scale Digital Devices. 2020 11th International Conference on Information and Communication Systems (ICICS), 2020, pp. 72–78, doi: 10.1109/ICICS49469.2020.239531.

[19] Li, S.—Choo, K. K. R.—Sun, Q.—Buchanan, W. J.—Cao, J.: IoT Forensics: Amazon Echo as a Use Case. IEEE Internet of Things Journal, Vol. 6, 2019, No. 4, pp. 6487–6497, doi: 10.1109/JIOT.2019.2906946.

[20] Kang, S.—Kim, S.—Kim, J.: Forensic Analysis for IoT Fitness Trackers and Its Application. Peer-to-Peer Networking and Applications, Vol. 13, 2020, No. 2, pp. 564–573, doi: 10.1007/s12083-018-0708-3.

[21] Zhou, H.—Deng, L.—Xu, W.—Yu, W.—Dehlinger, J.—Chakraborty, S.: Towards Internet of Things (IoT) Forensics Analysis on Intelligent Robot Vacuum Systems. 2022 IEEE/ACIS 20th International Conference on Software Engineering Research, Management and Applications (SERA), 2022, pp. 91–98, doi: 10.1109/SERA54885.2022.9806735.

[22] Raspberry Pi Foundation: Raspberry Pi 3 Model B. 2024, https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[23] Canonical Group: Ubuntu Core: The Embedded Linux OS for Devices. 2024, https://ubuntu.com/core.

[24] OpenWrt Project: Welcome to the OpenWrt Project. 2024, https://openwrt.org/.

[25] 504ensics Labs: 504ensicsLabs/LiME. 2022, https://github.com/504ensicsLabs/LiME.

[26] Pomeranz, H.: halpomeranz/Lmg. 2020, https://github.com/halpomeranz/lmg.

[27] Brune, N.: NateBrune/Fmem. 2022, https://github.com/NateBrune/fmem.

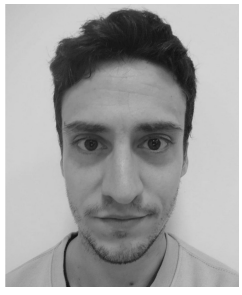
[28] Tcpdump/Libpcap Public Repository. 2024, https://www.tcpdump.org.

[29] Java, G. P.: IPTraf – An IP Network Monitor. 2024, http://iptraf.seul.org/.

[30] Warren, P.—Lightfoot, C.: iftop: Display Bandwidth Usage on an Interface. 2017, http://www.ex-parrot.com/pdw/iftop/.

[31] Kirby, C. et al.: HewlettPackard/Netperf. 2022, https://github.com/HewlettPackard/netperf.

Juan Manuel Castelo Gómez joined the Computer Architecture and Technology research group at the Albacete Research Institute of Informatics in 2016. In 2017, he received his M.Sc. degree in computer science from the University of Castilla La Mancha (Spain), and in 2021 he obtained his Ph.D. in advanced information technologies from the aforementioned university. His research interests are related to cybersecurity, especially digital forensics, as well as malware detection.



José Roldán-Gómez obtained his Bachelor's degree in computer engineering from the University of Castilla-La Mancha in 2017, his Master's degree in computer engineering from the University of Castilla-La Mancha in 2018, and his Ph.D. in advanced information technologies from the University of Castilla-La Mancha in 2023. His main interests are artificial intelligence applied to threat detection in IoT environments and automatic rule generation in CEP engines.

Sergio Ruiz-Villafranca received his B.Sc. degree in computer engineering from the University of Castilla-La Mancha in 2017 and his M.Sc. degree in computer science in 2021. Since then, he is a Ph.D. student in advanced information technologies at the aforementioned university. His research interests are related to cybersecurity, especially Industrial Internet of Things, machine learning for anomaly detection as well as computer forensics.

Álvaro del Amo Mínguez obtained his Bachelor's degree in computer engineering from the University of Castilla-La Mancha in 2020. His main interests are related to IoT forensics.