Becoming JUDAS: Correlating Users and Devices During a Digital Investigation

Ana Nieto

***Abstract*— One of the biggest challenges in IoT-forensics is the analysis and correlation of heterogeneous digital evidence, to enable an effective understanding of complex scenarios. This paper defines a methodology for extracting unique objects (e.g., representing users or devices) from the files of a case, defining the context of the digital investigation and increasing the knowledge progressively, using additional files from the case (e.g. network captures). The solution includes external searches using *open source intelligence* (OSINT) sources when needed. In order to illustrate this approach, the proposed methodology is implemented in the *JSON Users and Devices analysis* (JUDAS) tool, which is able to generate the context from JSON files, complete it, and show the whole context using dynamic graphs. The approach is validated using the files in an IoT-Forensic digital investigation where an important set of potential digital evidence extracted from Amazon's Alexa Cloud is analysed.**

***Index Terms*— IoT-forensics, digital investigation, OSINT, JSON, data normalisation, Alexa.**

I. Introduction

NOWADAYS, it is common that different user-oriented devices and services are designed to generate a large quantity of logs, which are then processed either by the administrators or automatic tools, in order to identify specific problems affecting their systems. Unfortunately, the correlation of digital evidence from different sources as part of the same digital investigation, is not an easy task when the number of files to be processed grows significantly, a problem that is affecting the *Internet of Things* (IoT) Forensic scenarios [1], [2]. In addition, there are several issues that complicate the understanding and correlation of digital evidence, where data normalisation plays a critical role [3]. In order to truly comprehend the digital scene, the data must be processed based on a set of common criteria able to highlight what is really relevant to the digital investigation. However, the heterogeneity of the sources hinders the identification, classification and correlation of data. Even when there is a large set of digital forensic tools able to export the results to common formats, the interpretation of the fields used by the tools can change.

For example, a common practice in data exchange, to save the results of operations or to generate logs, is to use the *JavaScript Object Notation* (JSON) [4] format. The JSON format is based on two structures: a collection of name/value pairs and an ordered list of values, such as object, array, string, number, and special values such as *true*, *false* and *null* [4]. This format is widely used by many applications, and also by services that are accessed using *Application Program Interface* (API) requests. The quality and quantity of data stored in a JSON, as well as the *tags* used to describe the values, depend on the application and the intended use of the JSON; therefore, these files can be very different in form and content. Some systems, such as Amazon's Alexa Cloud, can store important digital evidence in JSON files, according to [5] and [6], mainly describing the operations performed by the devices in such a context. Relevant digital forensic tools use this format. For example, the framework Volatility for memory analysis can provide the results of the operations executed as JSON files. In addition, there are modules in Python that can list the content of PCAP files as JSONs, including the IP and MAC addresses and all the data available. All these, once processed, can be useful to complete the information about a system. However, although there are multiple tools able to provide JSON files, there are no frameworks or solutions to integrate all this knowledge as part of the same context. This is a difficult task, considering that each tool can have its own notation to represent data.

Although there are solutions able to process some of these data, these have been developed for very specific purposes and usually do not consider integration with other tools to complete the knowledge. So it is difficult to progressively feed and improve the information collected during a digital investigation. It is also difficult to automate certain processes during the analysis without having to consider data normalisation, not only for a specific set of applications and devices, but also for new ones when they are added.

A. Motivation and Structure

As defined in [3], data normalisation is *the combining of evidentiary data of the same type from different sources with different vocabularies into a single, integrated terminology that can be used effectively in the correlation process*. This important requirement cannot be directly addressed given the heterogeneity of the sources and applications in the current *Information Technology* (IT) ecosystem. One of the motivations of this paper is to provide a methodology able to simplify the normalisation of data from different sources so as to identify relationships between users and devices during an analysis, in the context of a digital investigation. The JSON format is chosen as the starting point of the analysis because

Manuscript received April 18, 2019; revised October 15, 2019 and March 14, 2020; accepted April 14, 2020. Date of publication April 17, 2020; date of current version May 5, 2020. This work was supported in part by the Spanish Government through the Project IoTest under Grant TIN2015-72634-EXP/AEI, in part by SMOG Project under Grant TIN2016-79095-C2-1-R, in part by the EU H2020-SU-ICT-03-CyberSec4Europe (cybersec4europe.eu) under Grant 2018830929 and in part by INCIBE. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Lorenzo Cavallaro.

The author is with the Computer Science Department, University of Malaga, 29071 Málaga, Spain (e-mail: nieto@lcc.uma.es).

Digital Object Identifier 10.1109/TIFS.2020.2988602

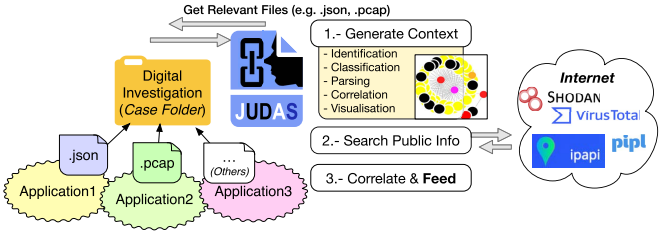

Fig. 1. JSON Users and Devices AnalysiS (JUDAS).

it provides us with a common umbrella under which we can identify common characteristics in the output of applications and API requests, being used for: i) identifying keywords and equivalences between different sources, ii) performing new requests for data that will be provided in the same format and iii) integrating the new data in the common framework defined in this paper.

With respect to large scale IoT-Forensic scenarios, thousands of logs and files have to be analysed after the acquisition. This approach considers the analysis of JSON files in order to identify the relevant fields to classify the data, providing a new abstraction layer for the digital investigator. Then, additional files can be analysed to include new data in the context. External services can also be used to improve the analysis, whenever possible. Therefore, the solution focuses on the analysis of digital evidence after it has been acquired.

The general idea is shown in Fig. 1. The solution proposed here takes into account digital evidence extracted from the different devices/systems involved. The JSON files are analysed to identify traces about devices (e.g., mobile phones, sensors or others) and users (e.g., account identifiers), and this information is visually shown in an interactive graph, representing the live context. The rest of the documents in the case can be processed and correlated with the current live context. In addition, this local information is enhanced using Open Source Intelligence Tools (OSINT) [7] services. The methodology is implemented in the *JSON Users and Devices analysis* tool (JUDAS), available at GitHub [8].

The structure of the paper is as follows. Section II describes related work. Section III defines the *JSON Users and Devices Analysys* (JUDAS) methodology, which is validated in Section IV using the developed tool of the same name. The experiments were performed using a dataset from the latest DFRW challenge [9], where the Alexa ecosystem is part of the scene. Finally, conclusions and future work are discussed.

II. State of the Art

Two interesting contributions directly related to the approach presented here are [5] and [6]. In [5] a *Cloud-based IoT Forensic Toolkit* (CIFT) is proposed to analyse Amazon Echo artefacts. The soluton combines cloud forensics and client-side forensics so as to have different sources of data to prevent relevant data from being lost (e.g. cloud-native artefacts). The authors define the Amazon Alexa ecosystem as the system created by all the interconnected devices used to customise the Alexa environment. The list includes: Alexa-enabled devices such as Echo, compatible IoT devices and third-party applications and companion clients, and those devices in which, although there are no specific applications for Alexa installed, it is accessed through a browser to configure the options. The structure of JSON files used by Alexa are detailed, because these are important digital evidence about the activity of the Alexa ecosystem. The description provided by the authors is very useful to better understand some specific tabs. More recently a solution for IoT-Forensics was proposed in [6], considering Amazon Echo as a use case. The authors propose a model to identify, acquire, analyse and present potential digital evidence. The solution focuses on the Amazon Alexa environment. The approach is very interesting and can help investigators conduct a digital investigation in these scenarios, contributing to the development of solutions for IoT-Forensics.

Indeed, several approaches in the context of IoT-Forensics have emerged since 2012 (aprox.). Some recent examples are [10], [11] and [12]. All of them stress the important change of context introduced by IoT devices and the new challenges to be addressed: problems of density, lack of resources and privacy issues all affect the acquisition and analysis of potential digital evidence. One of the main issues is the increased volume of data to be analysed, which complicates the interpretation of results.

Unlike the aforementioned contributions, this paper provides a solution for building an incremental representation of a digital investigation where the focus is on users and their relationships with a set of devices. The methodology proposed takes advantage of the tools and external services using JSON files. This format is used as a common representation for the input of data into the JUDAS system. The steps to be taken to identify, classify and parse the data considering the main objective of correlating users and devices are given. Thus, the files to be processed can be logs (e.g. from Alexa), results of local applications or even external services accessed using API requests (e.g. OSINT).

III. Methodology to Build the Context

The JUDAS methodology is defined, considering five phases (c.f. Fig. 2): identification, classification, parsing, correlation and visualisation, with the concept of *feeding* described as part of the correlation with *external* sources. The first three phases define the criteria used to normalise the data in JUDAS.

In Fig. 2 the phases are described, with an example, where two JSON files are provided as input: $J1$ and $J2$. For the identification, the content of the files is analysed in order to identify relevant keywords. The definition of keywords as identifiers helps classify the data into different objects in the second phase. In the classification the *objects of interest* are defined, based on the analysis performed during the identification and additional criteria. Thus, for example, considering just the tags in $J1$ and $J2$ it would be natural to deduce that only two types of objects can be defined to classify the data in our context, in this specific example: users and devices. However, additional classes can be added to prepare the system for analysing other targets, much more specific ones (e.g. a new class *Car* which inherits from *Device*.

After the classification, all the objects must be parsed, in order to synthesise the data in unique objects. One important

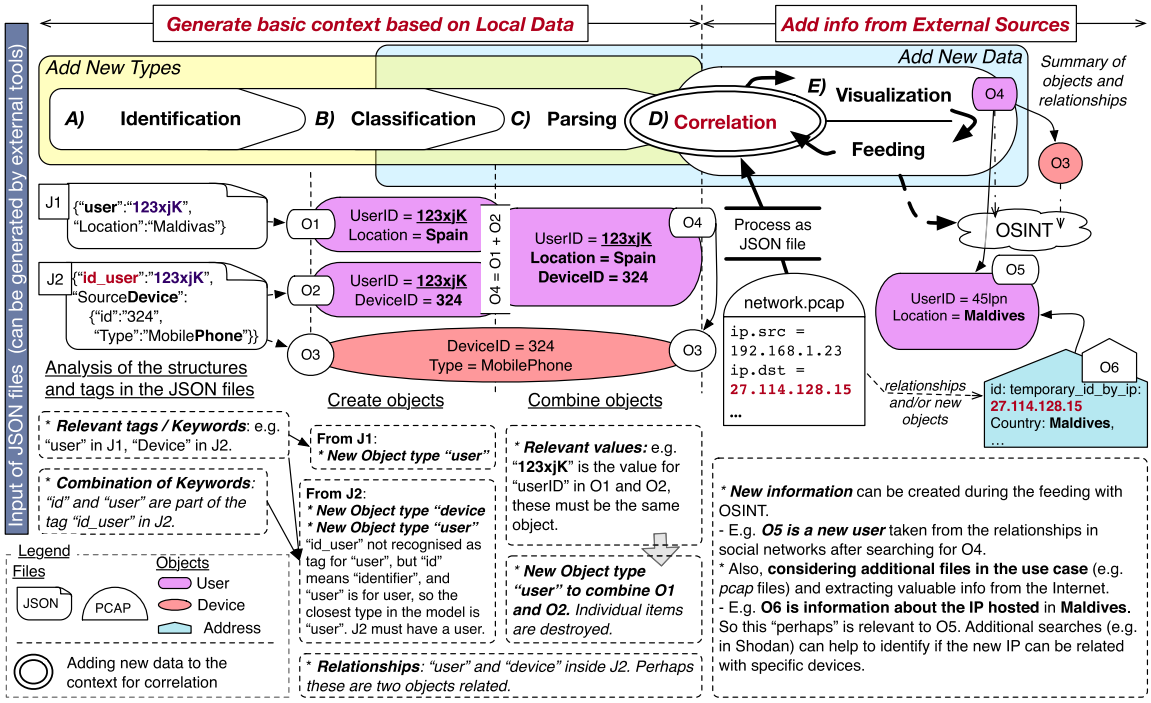

Fig. 2. Phases affected when adding new types (left) or data (right). New types require the modification of the schemas considered for identification, classification and parsing. However, new data only requires the modification of those phases where the data must be correlated (parsing and feeding).

part in this phase is the internal correlation, which is described in more detail in Section III-D, and is part of the process of data normalisation. In Fig. 2 $O1$ and $O2$ are summarised in $O4$, which is related to $O3$. Therefore, after the parsing, only two objects are included in the context: $O4$ and $O3$, which can be visualised using the graphical view.

At this point new data can be added to the context, using additional files in the use case. In Fig. 2 the file *network.pcap* is added to the context by analysing the data inside the PCAP and sending external requests to the OSINT services. The API requests for the IP return new JSON files following similar structures to Listing 1. In this example, there is a public IP 27.114.128.15 the location of which is the same as the user described in $O4$, so this can be relevant to the context. A new object of type Address is then included in the context.

Listing 1. Example: *ipapi* response for IP=27.114.128.15

```
{'ip': '27.114.128.15',
'city': 'Mal\'{e}',
'region': 'Kaafu␣Atoll', 'region_code': '26',
'country': 'MV', 'country_name': 'Maldives',
'continent_code': 'AS',
'in_eu': False, 'postal': None,
'latitude': 4.1667, 'longitude': 73.5,
'timezone': 'Indian/Maldives', 'utc_offset': '+0500',
'country_calling_code': '+960', 'currency': 'MVR',
'languages': 'dv,en', 'asn': 'AS7642',
'org': 'DHIVEHI␣RAAJJEYGE␣GULHUN␣PLC'}
```

Note that this means that in the classification a new type for *Address* must be considered, and also new methods must be added to identify similar objects of this class during the parsing. For this reason any change made to add new types will impact on the first phases of the methodology.

Last but not least, note that it is possible to increase the information available in the context without having to consider new files. For example, $O4$ can be used to deduce new information about the relationships of the user described with other individuals. The phases for visualisation and correlation (either internal or external) are repeated while new data is added to the context.

In this section the phases are described in detail, using specific examples considering the dataset analysed in Section IV. Note that the classes have been chosen taking into account all the information that has to be represented in this context. The methodology is defined in order to highlight common steps that have to be done to include new data. In addition, it is important to note that the steps defined are implemented in the JUDAS tool to carry out the whole process automatically.

It is important to note that the current solution is defined to help in the analysis of digital evidence that has been acquired in a previous step. In an investigation it may be the case that some artefacts are related to a malware campaign but not necessarily so. Note that the focus is on identifying users and devices. Of course the devices can be part of a botnet, and this can be identified during the analysis, but this is not always the case or the main purpose of this tool.

A. Identification

This phase is dedicated to understanding the structure of the files that will be processed by JUDAS, and is decomposed into two parts: identifying equivalent terms inside the files and identifying the relationships based on the context. The main terms in the JSON files have been identified and are ready for classification, once this phase is complete.

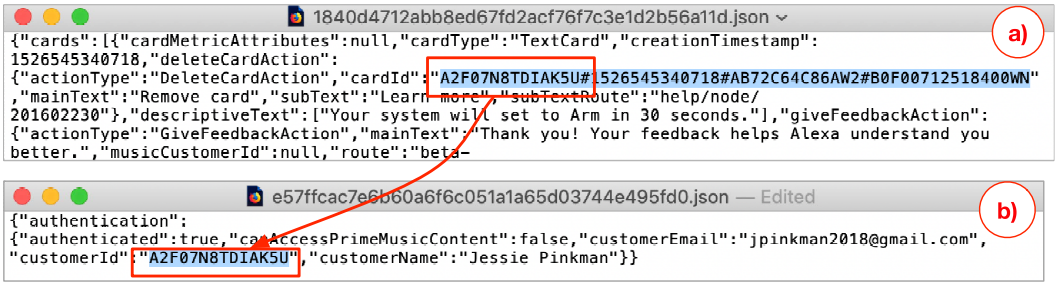

Fig. 3. Cards (a) and Authentication data (b). The same string is identified in two different JSONs in different ways. In (a) the string is part of the card identifier and in (b) is a identifier by itself.

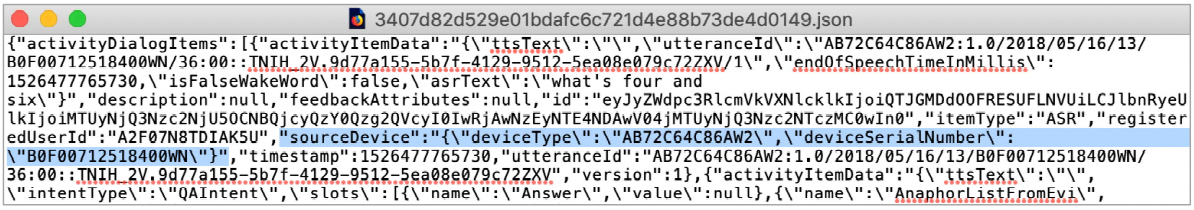

Fig. 4. This figure shows part of a JSON for Activity data, where the description for a device inside the Activity is highlighted.

TABLE I

EXAMPLES OF EQUIVALENCES IDENTIFIED IN THE JSONS

Label	Key in JSON (Equivalences)
User(id)	user, customerId, searchCustomerId, registeredCustomerId, registeredUserId, deviceOwnerCustomerId
Device(id)	device, sourceDevice, deviceSerailanNumber, deviceId, serialNumber
Card(id)	cardId
Type	type, cardType, itemType
PostalCode	postalCode, postal, Zipcode
Timezone	timezone, timeZoneId, timeZoneRegion
Timestamp	timestamp, creationTimestamp

1) Identify Equivalent Terms Inside the Files: Considering the heterogeneity of JSON files (c.f. Section I), it is critical to determine the mapping between the different keys used to access the values. For example, in Table I some equivalences identified in the writing of this paper are listed. The first column corresponds to the label used by JUDAS to represent the terms that are equivalent (second column).

The words *device*, *sourceDevice*, *deviceSerialNumber*, *deviceId*, *serialNumber* correspond to identifiers for devices. So if any of these words are included in a JSON, then it means that an object can be generated to represent a device. The same occurs with the words *user*, *customerId*, *searchCustomerId*, *registeredCustomerId*, *registeredUserId*, *deviceOwnerCustomerId* for users. In addition, there are other words that although they are not identifiers, can be mapped to the same concept for their interpretation. For example type can be expressed as *type*, *cardType*, *itemType* and so on, depending on the set of JSONs.

Furthermore, there are other types that are not considered by default and the system must be able to add them as new categories. The approach followed in this solution is to identify a set of pre-defined equivalences, principally for identifiers, not just those already mentioned, but also other identifiers about concepts (e.g. *timezoneId*). These words are also included in a list of keywords that belong to a dictionary, defined for each object in the system. In addition, during the processing of the JSON files, the list of keywords can be completed with new words identified as part of the context. Therefore, the system must be reviewed carefully when new JSONs with different formats are processed, so as to identify new equivalences (e.g. principally identified because different keywords have the same value), and added to the structure.

2) Identify Direct and Indirect Relationships Based on the Content: It is very important to identify the relationships between parts of the same JSON. This will help determine the initial relationships of the objects. This must be done considering: i) whether or not the structure analysed contains identifiers of other objects as part of its description, and ii) if any items contain other items.

The first case means that particular attention must be paid to those fields that can be composed by multiple identifiers of other objects. For example, Fig. 3 (a) shows the identifier for a *Card* ($cardId$). As can be seen, this is composed of different words, separated by the delimiter #. If we consider another file with authentication data (c.f. Fig. 3 (b)), it is possible to observe that the first word $A2F07N8TDIAK5U$ corresponds to a user's identifier. Therefore, the system should be able to understand that the card and the user are related. Furthermore, the words $AB72C64C86AW2$ and $B0F00712518400WN$ represent the type of device used and the identifier of the device, respectively.

The second case means that it must be identified when one item can contain others by the explicit definition of the same. For example, Fig. 4 shows data about an activity item, which, in this case, contains fragments of a conversation. If we observe this figure, then it is possible to identify the word *sourceDevice* with values that describe the type and identifier for a device. When this occurs, the decision taken is to extract this information from the main object (activity in this example)

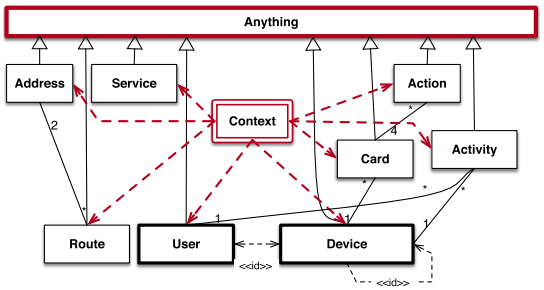

Fig. 5. Classification. The number of classes can increase if new entities must be represented. Classes *Anything* and *Context* must be maintained.

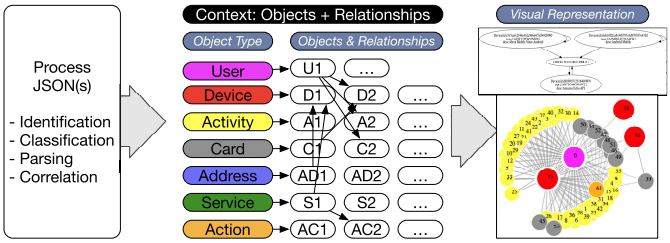

Fig. 6. Parsing, classification and visualisation of results.

but leave behind a trace of the relationship of the main object with the object extracted (device in this example).

Therefore, during this phase, the main items needed to describe the context are identified. After that, it is critical to define how to classify the type of information that will be deduced from the sources.

B. Classification

Once the aforementioned relationships are known, the next step is to identify the data that is independent from the rest and then define a set of basic types or classes to represent the *context*. In order to provide a complete solution, it is very important to take into account the requirements extracted from the previous phases, in particular what objects can potentially be contained in others, and also the fact that some objects can have different data depending on the type of file in which they are defined. Moreover, in this phase it is essential to select a representative set to classify all the data acquired from the files and to store new data during the progress of the digital investigation. It is also critical to define the possible dependencies between classes.

For example, Fig. 5 shows the classes defined according to the analysis of evidence in our case study: User, Device, Activity, Card, Address, Service and Action. The reason is because when analysing the JSON files there are several items that need these specific structures to define a specific behaviour or to differentiate the type of data that they store. The class *Anything* is a special class defined with all the default behaviour for any object added to the context. This class defines, for example, how the other classes print their data, the colour used to represent the objects of the classes and how new data can be added to the objects. All the classes in the proposed solution inherit from *Anything*, and the specific behaviour of a class is defined depending on the purpose and characteristics of that class.

Furthermore, some classes are more generic than others, so are useful for several contexts. For example, the classes *User* and *Device* intuitively serve to capture the data from very different files. In other words, while all the previously listed classes allow the representation of JSONs from an Alexa ecosystem, the object *User* is also useful to represent a user of the system described in an external JSON (not only in the Alexa context). The class *Address* can be used to represent the information about public IPs returned by the module *ipapi* (c.f. Listing 1) or the additional APIs used to search information about IP addresses, as well as the information about malicious IPs (if any) returned by VirusTotal.

As a further requirement, the definition of these classes needs to be as general as possible, so as to embrace the integration of very different items in the same context. It is even possible to discern the information that is natural from each sub-environment in the same digital investigation or identify the new objects generated due to interaction with the external sources, because each class can be fully customised (e.g. different colours for different devices).

Finally, this phase is closely related to the visualisation; if the classification is poor, then the rest of the phases may be affected. The parsing can be complicated, because the parser might not know where to place or discard the data, something which could also be the case in the correlation. Moreover, and perhaps more importantly, we may not be able to see where our failures are because the visualisation of the data strongly depends on how they are classified.

C. Parsing

In JUDAS the initial seed or state representing the context of the digital investigation is prepared during the parsing phase. To do this, all the relevant files must be analysed to collect all relevant data, interpret them and create the objects according to a previous classification. This phase is highly dependent on the classification. In the solution proposed, each class defines the way in which the data from JSON files must be interpreted to create the objects. During this phase the relationships between the specific objects are also identified. This corresponds to the scenario exemplified in Fig. 6, under the label *Objects & Relationships*, using white boxes and arrows. This is the initial knowledge that can be extracted from the selected files.

As stated, all the information about an individual, device or item is represented by the same, unique object. Returning to the initial example of the user with identifier $A2F07N8TDIAK5U$ (c.f. Fig. 3), this restriction means that when the parser finds the identifier of the user for the first time it generates an object with the identifier as the key. All the relationships with the user will leave a trace with this identifier. When the parser is able to extract another object from a different JSON file, if the identifier is the same, then the original object eats the second one to homogenise and complete the information, and the second object is discarded. This process is denoted *cannibalism* in JUDAS, and is described using the pseudo-code shown in Algorithm 1.

The preceding phase of classification allows the expected relationship between the objects to be known; for example,

Algorithm 1 Updating the **context** by Processing All the Objects, Ensuring Unique Objects and Identifying Relationships

```
1: procedure CANNIBALIM
2:     keys = getKeys()          ▷ List of all types in the context
3:     allvalues = getall()      ▷ All objects in the context
4:     mod = False
5:     for k in keys do
6:         valuesk = getall(k)
7:         newval = []
8:         for v in valuesk do
9:             for o in allvalues do
10:                done = v.eat(o)
11:                mod = mod or done
12:            newval.append(v)
13:        context.update({k : newval})
           return mod
```

Algorithm 2 Combine Two Objects That Are Equal; *backpack* Is a Dictionary, Used to Store the Description of Values

```
1: procedure EATSAME(object, destroy=False)
2:     if object is not equal to this thenreturn False
       ▷ 1. Combine descriptions
3:     keys2 = object.getKeys()
4:     eat = False
5:     for k in keys2 do
6:         values = getBackpack()
7:         values2 = object.getBackpack()
8:         for v in values2 do
9:             if v not in values then
10:                values.append(v)
11:            backpack[k] = values
       ▷ 2. Destroy the object if same
12:    if destroy then: object.delete()
```

it is expected that an Activity shows relationships between users and devices, allowing them both to be related to each other (c.f. Fig. 5). Then, the parser must consider these relationships to extract the data of users and devices while it is creating the Activity objects. JUDAS does not use a single parser. Rather, each class defines its own parser to improve the detection of objects following the requirements detailed during the identification (c.f Section III-A). When one object is identified inside another (c.f. Fig. 4), then the object is extracted by the parser and only the identifier remains in the parent. Additionally, a trace about the relationships between the objects (e.g., User, Device and Activity in the example) is grabbed in all the objects. However, in the same system, multiple objects can have this behaviour. Moreover, sometimes the information in the Activity is not complete, and the Activity only has the user identifier.

The parser is also able to make correlations between objects of different types based on the classification in the previous phase. Therefore, the operation *cannibalism* described in Algorithm 1 is also applied to objects of different classes in order to complete the information about common aspects that can affect various objects, such as the *address*. This operation is done during the parsing, while the objects are added to the context, and can be time-consuming. For this reason the programming language chosen to implement this phase is very important, in order to ensure efficient solutions.

While the parser is able to generate objects and complete the information about these objects, additional steps must be taken to complete the knowledge of the environment.

D. Correlation and Feeding

The correlation can be divided into two phases: internal correlation between the objects in the context, produced by the method *eat* during the parsing, and the correlation produced by external inputs. The latter is denoted *feeding* in Fig. 2 in order to highlight that it is a subsequent phase executed only when the context cannot grow, based on the data available in the local domain (folder) of the digital investigation.

Internal correlation is based on the inheritance. All the classes representing a type of object inherit from the class

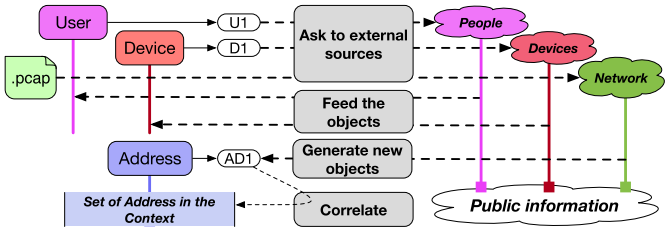

Fig. 7. External correlation for objects (user, device) and files to be processed.

Anything (c.f. Fig. 5), where the initial behaviour of the method *eat* is defined. *eat* depends on specific methods declared as abstract, which are different depending on the child implementing the specific behaviour. When the object to be eaten has the same identification than the object processing the call, the result of *eat* is a call to *eatSame*, described in Algorithm 2, common to all classes.

During the *external correlation* or *feeding* the initial information is completed with external information acquired from OSINT services. Note that JUDAS will determine the initial relationships in the same context based on the unique identifiers, identified during the parsing. Depending on the object this can be relaxed, including additional fields in the comparison, such as the users' email addresses. For example, two objects with different identifiers but the same email can be interpreted as the same, but the list of names is maintained in case these are in fact different, and this can be identified in the visualisation of results.

In Fig. 7 the alternatives implemented in JUDAS for *external correlation* are described. First, there are some objects with fields that can be useful for automatic searches in public, the external sources, named previously (e.g. pipl). For example, the fields *name* and *email* for a user can be used to get additional information about the user. The requests for the external tools are generated based on the available information in the objects and the responses of the external services (if any) are added to the current objects in the context.

In addition, it is possible to search for additional information that is present in other types of files (not JSON). For example, when analysing network files (.pcap), it is possible to extract the IPs and MACs found in the file. Public IPs are then selected and information about them is requested from external sources.

Note that, in this case, there are no objects representing the IPs in the context. Therefore, only when the external services provide their results about the IPs can new objects be generated, in this case objects of type *Address* are used. Then, the content in this new object is compared with the rest of the objects of the same type in the current context. In this case the comparison cannot be done using the identifier directly (because new objects have a temporary *id*, c.f. Section IV-B), so the correlation defines the matches based on the postal code of the addresses and also the zone identifier. The pseudo-code shown in Algorithm 3 describes the method for generating new objects based on the information collected from public IPs using the module $ipapi$. The result of this algorithm is used in Algorithm 4 to perform the correlation with the objects.

Algorithm 3 Request Information About Public IPs Using *ipapi* and Generate New Objects of Type Address

```
1: procedure GETINFOIP(iplist)
2:     addresses = []
3:     for ip in iplist do:
4:         if ipinfo.ispublic(ip) then:
5:             info = ipapi.location(ip)
6:             jsonarray = json.dumps(info)
7:             a = json.loads(jsonarray, object_hook =
                   eatjson.Address.as_address)
8:             if a is instance of Address then:
9:                 addresses.append(a)
       return addresses
```

Algorithm 4 Correlating New Addresses With the Current Context. This Version Searches for Matches in the Timezone and the Postal Code, but Additional Matches Can Be Defined

```
1: procedure CORRELATENETADDRESS(addresses)
       ▷ 1. Get devices with addresses defined
2:     mydevs = getall('Devices')
3:     mydevs = remove devices without addresses from mydevs
       ▷ 3. Check for matches
4:     for d in mydevs do:
5:                          ▷ 3.1. Get the ID of the addresses (strings)
6:         addevlist = d.getBackpack('Address')
7:                                  ▷ 3.2. Get the objects for the IDs
8:         addevlist = getObjects(addevlist,'Address')
9:                  ▷ 3.3. Matches timezone (mtz) and zipcode (mzc)
10:        mtz = list of pairs [apub, adev] where apub is
              in addresses and adev is in addevlist and
              apub.timezone() == adev.timezone()
11:        mzc = list of pairs [apub, adev] where apub is
              in addresses and adev is in addevlist and
              apub.zipcode() == adev.zipcode()
       ▷ 4. Process results
12:    if len(mtz) + len(mzc) > 0 then:
13:        results = write results friendly
14:        Add input to the history of the context
       ▷ 5. Return results return results
```

The new data, provided by the external sources, can be included or not in the context. This part is easily modifiable to include as many sources of information as desired.

E. Visualisation

The visualisation of the results is a critical part of providing useful tools. JUDAS defines this phase following two requirements: i) separate the context from other tests/searches over the digital evidence, ii) simplify the visualisation of results as much as possible while still providing a general view.

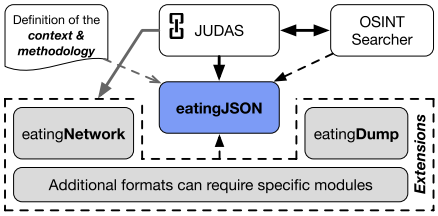

Fig. 8. JUDAS - Modules developed to the proof of concept.

The first decision made is to separate the visualisation of the context and the operations on it from the operations on the files containing the digital evidence (JSON files and other files used to generate the context). This is required because the context is a representation of all the data, but during the normalisation some information can be misunderstood or missing. Therefore, the description of the objects generated is shown in the main window, as shown in Fig. 9. Keyword searches can be done on both the objects of the context and on the source files.

The second requirement emphasises that although the data of the context are numerous, we must find a way to express them in their entirety so as to have a complete vision of all the relationships. The solution proposed implements this requirement using interactive graphs, generated using *networkx* for Python and additional technologies to plot the graph in the browser of the investigator. In Fig. 10 an example of the results following this premise is shown. Each circle is a different object in the context. The colours represent the type/class of the object, and the numbers are identifiers generated to simplify the visualisation of results (e.g. some identifiers are very large and muddle the view). Additional information about the devices is shown when needed by clicking on the object.

IV. PROOF OF CONCEPT: JUDAS TOOL

The JSON Users and Devices AnalysiS (JUDAS) tool has been implemented to validate the methodology. It is able to process and extract relevant data, concerning users and devices from JSON files. After parsing the data from the files, the tool is able to make a basic correlation of the items identified. Then, the digital investigator can ask for specific inputs based on different criteria (e.g. keywords found as identifiers).

JUDAS has been implemented using Python 3 to take advantage of the multiple modules to aid in processing JSON files and other formats. The methodology is implemented in the central module named $eatingJSON$, shown in Fig. 8, while the files with different formats (e.g. network files or memory dumps) are pre-processed in additional modules developed for the proof of concept. All the modules must be developed taking into account the methodology and classes included in $eatingJSON$. The OSINT module is designed to search using the fields in the objects provided by the model. The methods are further documented in the public repository updated in GitHub [8]. In order to increase the functionality provided, for example including new types of JSON files to be understandable to the tool, only the hooks used during

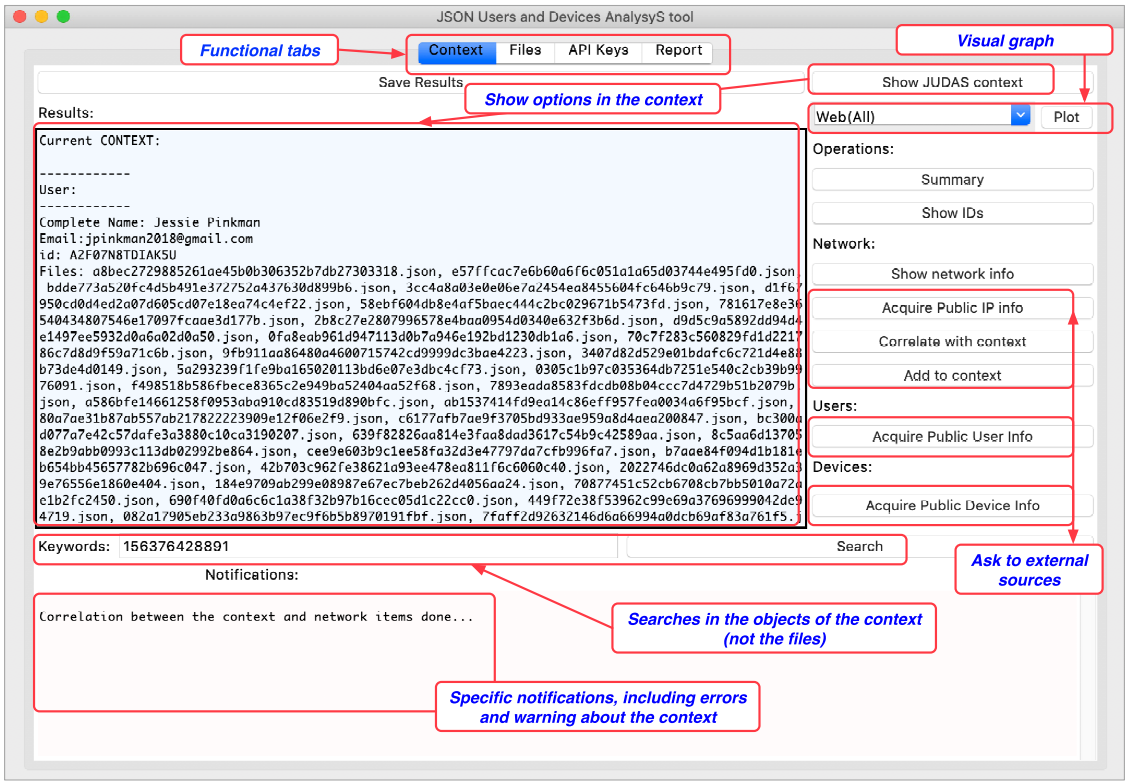

Fig. 9. Overview of the first window shown by the JSON Users and Devices AnalysiS (JSON) Tool, after loading the files of the case.

the parsing have to be modified for the corresponding new behaviour to be implemented, following the criteria described in the methodology (c.f. Section III). To include new external sources to check and correlate the results, it would be sufficient to modify the main interface of JUDAS (c.f. Section IV-A); however, depending on the source it could be useful to develop new classes to represent the new data (c.f. Section III-B).

The following sections describe the basic usage of the tool, some results after using it and the summary of statistics.

A. Quick Overview

JUDAS provides a *Graphical User Interface* (GUI) to simplify the analysis (Fig. 9). The first step is to select a default folder, where all the files of the same digital investigation will be allocated. There are four tabs at the top of the GUI: Context, Files, API Keys and Report. The first tab is for including all the operations that can affect the context (objects generated by the tool). The second tab is for selecting the default folder and also the type of files that can be added to the context (for the current tests, only .json and .pcap files are collected). The collection of relevant files is recursive. The third tab is for including API keys, which are needed to use the external services (e.g. Shodan). The last tab is for reporting. The actions applied to the object are included in the report to simplify the traceability of our actions in the system. These reports can be saved to aid in the writing of the final report of the case.

In addition, there are different options or buttons whose objective is to show the different features implemented following the premisses detailed in the methodology. Note that the text in Fig.9 shows all the information in the object, including

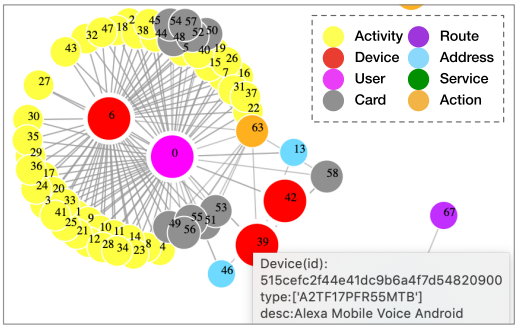

Fig. 10. Section of the dynamic graph generated for JUDAS for Alexa JSONs.

the files where all the data can be found. In addition, each object stores its own registry of its activity: time when it was created, the object from which it was generated/extracted (if any), etc.

B. Proof of Concept: Results From Amazon's Alexa

JUDAS is validated using a representative set of data downloaded from the DFRW 2017/18 challenge [9] closed in March 2019. These sources provide a rich context for analysis, where, for the sake of clarity, only one part is used to test JUDAS. The drawback to the dataset chosen is that both the users and the devices are fictitious. So, the services consulted for external information do not provide enough substantial information to complete some parts of the context.

In Fig. 10 the context generated for the files downloaded from the Alexa Cloud is shown. The first view of the circle suggests that device #6 and user #0 are related inside the

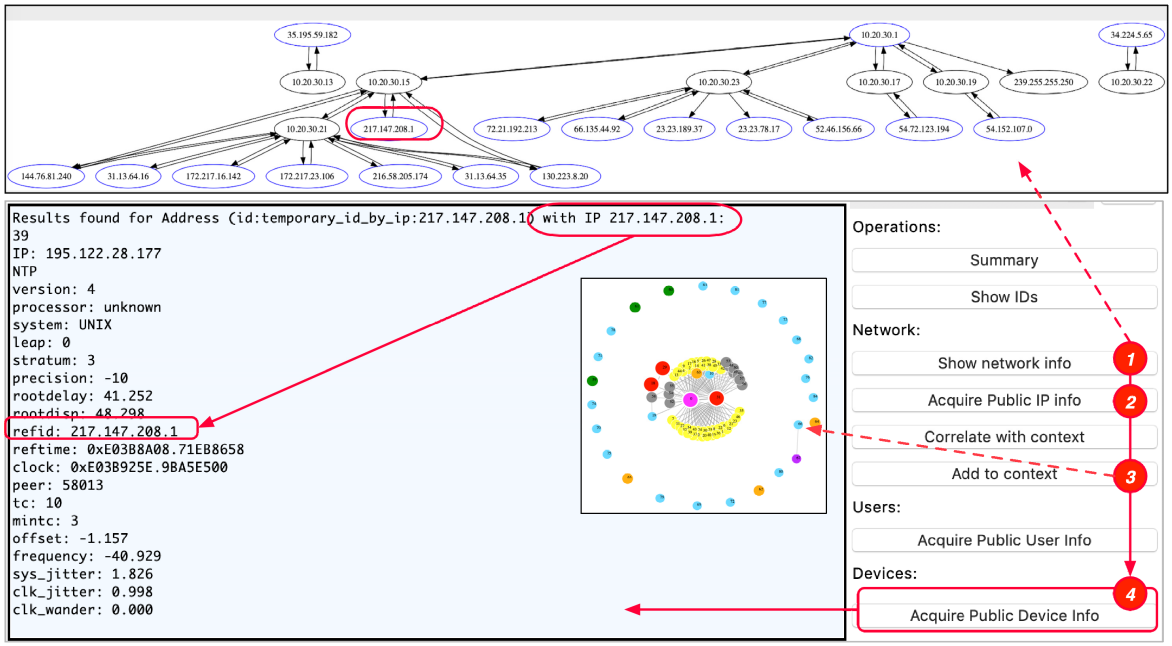

Fig. 11. Parsing a PCAP file to identify public IPs and check them using *ipapi* and Shodan. New Addresses can be added to the context after this step.

majority of objects of type Activity. These relationships are normal because #6 is an Echo device used to communicate with the Alexa Cloud. Unlike #6, devices #39 and #42 are related to the user, but their presence in the logs is not as prominent as #6. Analysing the characteristics of the devices, apparently these are mobile devices used either by the user or by other users in the system with the (apparent) authorisation of the only identified user, who is the owner of the Echo device. If both objects are analysed, the tool shows different account numbers that can be further analysed.

Moreover, the dataset used for this validation includes a network capture in a .pcap file. Using the option *Show network info* (step 1 in Fig. 11) the file is processed, and JUDAS shows a list of public and private IPs, the MAC address and two graphs to show the relationships between these values in the context. The first graph at the top of Fig. 11 shows the graph for IPs. Then, the option *Acquire Public IP info* (step 2) implements the steps shown in Fig. 7, using external services to acquire additional information for public IPs and creating new Addresses that can be added to the context (step 3). The new addresses are created with the "id" *temporary* and the IP address as keywords to differentiate these from the addresses with an identifier acquired from the JSON files. Then, once the new addresses have been added to the context (step 3), new items appear in the graph; there are no direct relationships with the user and devices because public IPs are not directly correlated. In this case, the addresses can be added to the context in order to search for additional information about the IPs in Shodan (step 4). For example, the arrow at the bottom in Fig. 11 leads to the results from Shodan for IP 217.147.208.1. In this case, the request returns a set of new IPs with the same *refid* parameter, which is the IP used in the search.

In addition, the option *Correlate with context* can be used to check if the information acquired about the public IPs (step 2) has any similarities with current addresses in the

Fig. 12. Output for correlation (*Timezone* and *PostalCode*).

context before adding the new data to it (step 3). Therefore, the option implements Algorithm 4, and can be called between steps 2 and 3, producing similar results to those shown in Fig. 12. However, note that using the dataset chosen, although some matches have been found, they are not strong enough, because the postal code for the addresses is not the same. The correlation across additional axes is possible by adding the desired fields in Algorithm 4. One of the future improvements planned for the tool is to be able to select the values using the GUI, avoiding the modification of the code.

C. Statistics

In addition to the features for working with the context, JUDAS provides some data about the number of objects that were created (number of instances of the classes), the number of instances that are finally used (unique instances), the number of relationships per class and the number of source files from where the instances were created. These data can be consulted using the button "Summary". In Fig. 13 the tool's output for the data analysed is shown. This graph is a summary, which demonstrates the nature of the files analysed and the analysis completed by JUDAS. The specific information about all the instances/objects finally used to define the context is shown in the main view (c.f. Fig. 9). For example, in Fig. 13 the output data for the class *User* means that, while the tool was analysing files, 231 instances of the class *User* were created by the tool after analysing 44 files.

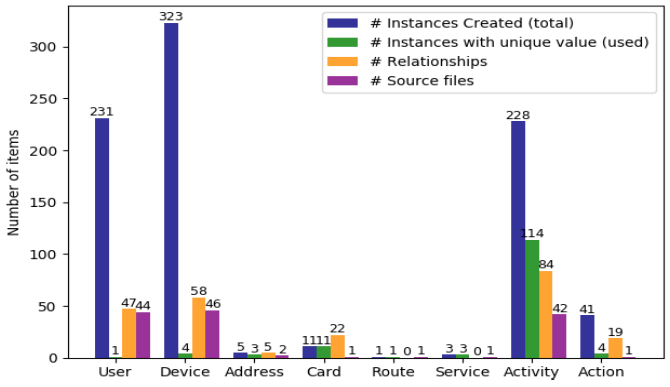

Fig. 13. Number of instances and relationships created by JUDAS.

However, all the instances are for the same unique user, so all the instances are eaten (c.f. Section III) in a single instance. Moreover, the unique instance is related to 47 instances (other instances of other classes) in the final state of the context. In addition, there are instances that are not connected with the user that can be omitted in this point of the analysis.

Note that all objects of the type *Action* are created from the same file. In this case, the summary does not include the external sources, and only shows those files used in the use case. If the same test were to be repeated after acquiring public IP info, then the number of instances for Addresses would be higher. In addition, intuitively, the results highlight that multiple instances of the same class with the same Id can be present in the same file.

In a real scenario, the parsing and correlation can be critical to simplify the identification of users and devices in unique objects. The tool summarises relevant words in a list of tags that can be used to compose wordlists to perform additional analyses improving the context. While the number of tags is not limited, JUDAS will be more efficient if the tags are mapped to the equivalences avoiding duplication.

Finally, these outputs are provided not only to reveal how many objects with the same identifier are created and combined, but also to contribute to improving the solution in future versions, and simplify the comparison with new, future work. Moreover, it helps ensure that all the appearances of the same object can be extracted from the files and no knowledge is lost. This also allows the impact of these objects on the performance and the cost of processing to be seen. As the tool becomes more complex, it will require new metrics to evaluate performance for different uses cases.

V. Conclusions and Future Work

This paper has proposed the *JSON Users and Devices analysis* (JUDAS) methodology to correlate users and devices, taking advantage of the JSON format widely used by many tools, either to save logs or to provide results of operations on data during the analysis of digital evidence. JUDAS generates a unified representation of the context of a digital investigation using the data available in the use case folder, and asks external services to complete the information about the objects generated. In addition, following the steps defined in the methodology, a tool with the same name has been developed to allow the integration with different tools and modules able to simplify the parsing of new formats to be added to the solution. The context draws from JSON files, but alternative methods can be included in the future. The next steps will focus on increasing the number of files that can be processed (improving the classification and parsing) and also the number of external sources that can be used to acquire public information to feed the context.

Moreover, *Natural Language Processing* (NLP) can be useful to improve the analysis of digital data concerning communication between individuals. Taking advantage of NLP, new modules focused on the interpretation of the text messages exchanged between users can be defined to provide additional information about the traits underlie personality. Considering that Alexa (and similar platforms) stores text speech, this can be very interesting.

Last but not least, the context is delimited by the files of a use case, or those belonging to a digital investigation. For the sake of simplicity a folder with all the potential digital evidence is considered, but this can be much more complex in the case the digital information is distributed over several sources. Additional tools for correlating data from different digital investigations without revealing the content of the data itself could be helpful to further complete the data acquired.

References

[1] E. Oriwoh, D. Jazani, G. Epiphaniou, and P. Sant, "Internet of Things forensics: Challenges and approaches," in *Proc. 9th IEEE Int. Conf. Collaborative Comput., Netw., Appl. Worksharing*, Oct. 2013, pp. 608–615.
[2] M. Conti, A. Dehghantanha, K. Franke, and S. Watson, "Internet of Things security and forensics: Challenges and opportunities," *Future Gener. Comput. Syst.*, vol. 78, pp. 544–546, Jan. 2018.
[3] P. Stephenson, "A comprehensive approach to digital incident investigation," *Inf. Secur. Tech. Rep.*, vol. 8, no. 2, pp. 42–54, 2003.
[4] D. Crockford. *Introducing Json*. Accessed: 2019. [Online]. Available: https://www.json.org
[5] H. Chung, J. Park, and S. Lee, "Digital forensic approaches for Amazon Alexa ecosystem," *Digit. Invest.*, vol. 22, pp. S15–S25, Aug. 2017.
[6] S. Li, K.-K.-R. Choo, Q. Sun, W. J. Buchanan, and J. Cao, "IoT forensics: Amazon echo as a use case," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 6487–6497, Aug. 2019.
[7] *Osint Framework*. Accessed: 2019. [Online]. Available: https://osintframework.com/
[8] *Json Users and Devices Analysis*. Accessed: Apr. 2019. [Online]. Available: https://github.com/cadirneca/judas
[9] *Dfrws Forensic Challenge*. Accessed: Mar. 2019. [Online]. Available: https://www.dfrws.org/dfrws-forensic-challenge
[10] M. Hossain, Y. Karim, and R. Hasan, "FIF-IoT: A forensic investigation framework for IoT using a public digital ledger," in *Proc. IEEE Int. Congr. Internet Things (ICIOT)*, Jul. 2018, pp. 33–40.
[11] C. Meffert, D. Clark, I. Baggili, and F. Breitinger, "Forensic state acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition," in *Proc. 12th Int. Conf. Availability, Rel. Secur. (ARES)*. New York, NY, USA: ACM, 2017, p. 56.
[12] A. Nieto, R. Rios, and J. Lopez, "IoT-forensics meets privacy: Towards cooperative digital investigations," *Sensors*, vol. 18, no. 2, p. 492, 2018.

Ana Nieto is currently a Post-Doctoral Researcher with the University of Malaga. In 2018, she was awarded with a grant to develop advanced cybersecurity research by the Spanish Institute of Cybersecurity (INCIBE). She is currently a Senior Member of the Network, Information, and Computer Security (NICS) Lab. She is also working on the definition of new solutions to deal with the new challenges for the IoT-Forensics, such as the definition of trustworthy entities in the figure of *digital witness* and new mechanisms to the definition of cybersecurity context in the scope of a digital investigation. Her research interest includes digital forensics.