8-29-2017

Forensic State Acquisition from Internet of Things (FSAIoT): A General Framework and Practical Approach for IoT Forensics through IoT Device State Acquisition

Christopher S. Meffert
University of New Haven

Devon R. Clark
University of New Haven

Ibrahim Baggili
University of New Haven, ibaggili@newhaven.edu

Frank Breitinger
University of New Haven, fbreitinger@newhaven.edu

Publisher Citation

Christopher Meffert, Devon Clark, Ibrahim Baggili, and Frank Breitinger. 2017. Forensic State Acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition. In Proceedings of ARES '17, Reggio Calabria, Italy, August 29- September 01, 2017, 11 pages.

Comments

Dr. Baggili was appointed to the University of New Haven's Elder Family Endowed Chair in 2015.

Forensic State Acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition

Christopher Meffert
University of New Haven, UNHcFREG
West Haven, Connecticut
cmeff1@unh.newhaven.edu

Devon Clark
University of New Haven, UNHcFREG
West Haven, Connecticut
dclar4@unh.newhaven.edu

Ibrahim Baggili
University of New Haven, UNHcFREG
West Haven, Connecticut
ibaggili@newhaven.edu

Frank Breitinger
University of New Haven, UNHcFREG
West Haven, Connecticut
fbreitinger@newhaven.edu

ABSTRACT

IoT device forensics is a difficult problem given that manufactured IoT devices are not standardized, many store little to no historical data, and are always connected; making them extremely volatile. The goal of this paper was to address these challenges by presenting a primary account for a general framework and practical approach we term Forensic State Acquisition from Internet of Things (FSAIoT). We argue that by leveraging the acquisition of the state of IoT devices (e.g. if an IoT lock is open or locked), it becomes possible to paint a clear picture of events that have occurred. To this end, FSAIoT consists of a centralized Forensic State Acquisition Controller (FSAC) employed in three state collection modes: controller to IoT device, controller to cloud, and controller to controller. We present a proof of concept implementation using openHAB – a device agnostic open source IoT device controller – and self-created scripts, to resemble a FSAC implementation. Our proof of concept employed an Insteon IP Camera as a controller to device test, an Insteon Hub as a controller to controller test, and a nest thermostat for a a controller to cloud test. Our findings show that it is possible to practically pull forensically relevant state data from IoT devices. Future work and open research problems are shared.

KEYWORDS

Internet of Things, IoT State acquisition, IoT forensics framework, IoT research, IoT controllers, IoT forensic challenges

ACM Reference format:
Christopher Meffert, Devon Clark, Ibrahim Baggili, and Frank Breitinger. 2017. Forensic State Acquisition from Internet of Things (FSAIoT): A general framework and practical approach for IoT forensics through IoT device state acquisition. In *Proceedings of ARES '17, Reggio Calabria, Italy, August 29-September 01, 2017,* 11 pages.

ARES '17, August 29-September 01, 2017, Reggio Calabria, Italy

ACM ISBN 978-1-4503-5257-4/17/08…$15.00
https://doi.org/10.1145/3098954.3104053

https://doi.org/10.1145/3098954.3104053

1 INTRODUCTION

The Internet of Things (IoT) has enabled the creation of numerous always Internet-connected smart gadgets for end users, e.g., toasters, refrigerators, thermostats, locks, washing machines, car garage doors or motion detectors that connect to online services and platforms. Having devices always connected produces new types of cyber-physical evidentiary data. Nevertheless, how to acquire forensically relevant data and how to analyze it from different IoT devices without a common interface, internal storage or standard protocols is a challenge. While these are current major challenges, the shift towards everything connected also bring new avenues for digital evidence and may pinpoint the exact date and time a door was opened / locked, the temperature change, or when a car was parked, which in turn could help find digital evidence of forensic value in a potential case.

In this article, we propose a primary account for a generalized framework, FSAIoT, for data collection from a variety of different IoT devices. More precisely, our objective was to construct a general, encompassing, practical, methodology to facilitate forensic collection of state data from a multitude of IoT devices. As the related work (Section 2) illustrates, there are technical limitations related to obtaining this data. While some IoT devices keep historical records, many do not or are limited in the amount they do keep. Notwithstanding, a number of IoT devices rely on some sort of feedback loop to provide a current device state or to monitor a state change. We contend that by monitoring these states and or state changes we can acquire forensically valuable data.

Our work provides the following primary contributions:

- We present a definition of a Forensic State Acquisition Controller (FSAC).
- We present an account for a state acquisition framework for IoT devices.
- We illustrate through experimental work the feasibility of this framework using openHAB[1], a vendor and technology agnostic open source automation software for one's home.

[1] http://www.openhab.org/

We begin this paper by stating the forensic challenges related to IoT forensics in Section 3. We follow that by describing our proposed FSAIoT framework in Section 4, followed by a proof of concept implementation of the framework in Section 5. We then discuss our findings in Section 6 and limitations in Section 7. Finally, we wrap up with our conclusions in Section 8 and discuss areas that need future work in Section 9.

2 RELATED WORK

The next subsections present research on wearable IoT forensics (Section 2.1), mobile forensics (Section 2.2) and IoT forensics (Section 2.4). Section 2.3 reviews logs and how they are used in digital forensics, and Section 4.1 discusses IoT controllers. It is of note that at the time of writing, literature on IoT forensics was sparse.

2.1 Wearable IoT forensics

At the time of writing, smart watches were the most widely adopted IoT wearable devices. [11] noted projected sales of 100 million smart watches by 2020. Given that smart watches are employed in everything from Global Positioning System (GPS) locations to finical transactions, they are a haven of potential digital evidence. Smart watches have been shown to have security vulnerabilities. For instance, [2] showcased methods for physically acquiring the Samsung Gear 2 Neo watch and the LG G watch, whilst exemplifying the forensic implications of the methods used. Many smart watches still require synchronization with a mobile device, so we argue that mobile device forensics is strongly relevant to IoT forensics.

2.2 Mobile device forensics

Mobile devices are classified as IoT devices. They started out as always connected, simple devices, and have now become hand-held interconnected computers. While nowadays most mobile devices are either Android or iOS devices, this was not always the case. [6] highlighted several other mobile operating systems. Some are outdated and no longer used such as Palm OS, and others remain obscure and rarely employed such as Samsung's BADA[2]. Mobile devices at some point did not only have different operating systems, they were also engineered with proprietary hardware and hardware connectors.

As major mobile players emerged, devices became more standardized. This made developing and engaging in mobile forensics less complex as both kernels and filesystems became well known [3].

We hypothesize that the same trend will appear in IoT devices as there is presently no standardized operating system for IoT. Currently, each manufacturer adopts different development platforms and operating systems and many hardware interfaces are also proprietary in nature. Nevertheless, one common thread that all IoT devices have in common are their states, which may be logged.

2.3 Logs

Given that most IoT devices have limited long term storage, logs are of relevance to their forensic acquisition and analysis. If we cannot capture disk-related data, or a device's internal memory, perhaps we can capture a device's state, and log it.

It is no secret that logs contain forensically valuable information. In [12], logs are defined as a regular or systematic records of events or state changes that have occurred over a period of time. [12]'s work suggests that logs are the most common data source used in digital investigations. In particular, their work uses logs to overcome the difficulties in cloud forensics.

Another area that logs play a role in is investigative profiling. [1] suggest that investigative profiling helps reduce noise when attempting to identify bad actors and the motivations of bad actors.

In work by [5], a finite state machine model is used to reconstruct an event. A finite state machine is defined as a graph with nodes serving as possible states. Arrows represent the state changes. Therefore by back tracing, one can determine all previous states. These state changes can be obtained from logs.

When constructing time lines from log files, there may be inconsistencies and or contradictory information. These inconsistencies may help in finding what happened on a system given that we know what should have actually happened, creating an anomaly. Work by [8] showcased a tool, Computer Activity Timeline Detection (CAT Detect), which was developed and implemented to assist in this problem. While logs are applicable to all sub-domains in digital forensics, they are especially important to IoT forensics.

2.4 IoT forensics

In [9], IoT was broken down into domains such as cloud services, visualization, mobile devices, fixed computing, sensor and RFID technologies, and artificial intelligence. This helped highlight differences between traditional digital forensics and IoT forensics. With differences known, they developed a model that consists of three zones. These zones help guide investigators where to begin their work with regards to an investigation. Zone 1 is made up of the internal network, Zone 2 consists of all devices and software on the edges of the network, and Zone 3 consists of any hardware or software outside of the internal network. This zone conceptualization assists practitioners in developing an organized investigative plan.

In [10], the idea of automated forensics is proposed through a concept they title Forensics Edge Management System (FEMS). The idea being to develop and implement an autonomous security and forensic service within a facility that has IoT devices placed in it. The FEMS architecture consists of three layers; perception, network, and application. The layers work as follows: the perception layer collects data from the sensors, the network layer manages the transmission of data between the perception layer and the application layer, and the application layer is the interface to the end user. FEMS will collect data and store it for a designated period of time. This recording process is defined by an event that crosses a set of predefined thresholds. Although some of their concepts are applicable to our work, their work was theoretical in nature as they didn't dive deep into the state acquisition of IoT devices.

[13] defined IoT forensics as a branch of digital forensics, where the identification, collection, organization, and presentation processes deal with IoT infrastructures to establish facts about an incident. The work goes on to break IoT forensics into three different fields: Cloud, Network, and Device level forensics. They propose

[2]https://www.shoutmeloud.com/top-mobile-os-overview.html

a Forensics-aware IoT (FAIoT) model. Similar to FEMS described earlier, a centralized evidence collection point is suggested, allowing for ease of access and organization with regards to collected evidence. A secure evidence repository service is also proposed. An end user would register their devices to it allowing constant monitoring. All logs would be collected and separated according to what is collected in them. To facilitate ease of access to this data they also described an API service.

3 IOT FORENSIC CHALLENGES

With the dawn of the IoT age and advancements made in nearly every aspect of digital systems, we have reached a critical tipping point in the world of digital forensics. [4] points out that many of the tools and techniques that once worked without question are quickly becoming obsolete. File formats for storing forensically relevant data are becoming proprietary often requiring complex reverse engineering efforts. Data is often split into many elements and stored in the cloud. There are also legal challenges which limit data investigators are able to gain access to.

Harichandran et al. [7] noted that in the foreseeable future IoT is going to pose challenges to digital forensic examiners, yet, most research on IoT and how it relates to digital forensics has been mostly theoretical. It has also been indicated that IoT devices present a complex dilemma for digital forensic investigators due to the sheer number of different systems on the market. While few IoT devices may be acquired and analyzed using traditional digital forensic techniques, many are engineered with proprietary closed source software and file structures. Adding to the complexity, their communication protocols can be just as diverse, whether it be Bluetooth, WiFi, RF, ZigBee, etc.

Another major challenge is that many IoT devices employ Real-Time Operating Systems (RTOS) that serve real-time applications and process data as it comes in, typically without buffering delays. The processing times (including OS delay) are calculated in tenths of seconds or shorter increments of time[3]. This means that data is usually not stored in a RTOS making it difficult for examiners to forensically acquire digital evidence from IoT devices.

Overall, these challenges prevent investigators from being able to obtain and analyze IoT evidentiary data in a simple and timely manner. We argue that this is a result of practitioners continuing to chase historical data. The common idea of forensics being a post-mortem field where historical data is collected and analyzed, while still ideal, limits investigators in their work, especially when dealing with devices that have limited or no storage capacity. As part of this historical data, often, logs are analyzed to examine particular state changes. We argue that knowing the state change of an IoT device may be a practical solution to the IoT challenges at hand.

In an effort to overcome these obstacles, we devised and tested a primary approach for aggregating state data from IoT devices based on the acquisition framework we present in Section 4.

4 FSAIOT FRAMEWORK

The Forensic State Acquisition from Internet of Things (FSAIoT) framework consists of a centralized controller which we title Forensic State Acquisition Controller (FSAC, explained in Section 4.2) and three state collection methods where state refers to the current state of an IoT device (e.g. door open or closed). The three methods or modes are as follows: (1) controller to IoT device (see Section 4.3), (2) controller to cloud (see Section 4.4), and (3) controller to controller (see Section 4.5). Together, these three modes allow for the state acquisition of a multitude of IoT devices. In order to realize / implement the FSAC, we analyzed existing IoT controllers to provide a brief overview of existing solutions and motivate why we implemented our FASC using *openHAB* (Section 4.1).

4.1 IoT Controllers

IoT controllers provide a centralized mechanism by which multiple IoT devices can connect to, be managed and controlled. Through a controller, developers and engineers can support a variety of hardware and communication platforms. These controllers can typically acquire and modify the state of an IoT device affording users a single mobile application for controlling all the IoT devices in their home.

Besides usability, these controllers also have security benefits. For instance, [14] explained that there are pitfalls to having each individual device attempt its own connection to the cloud. Privacy and security are the top of the list. They propose a centralized approach by using small gateway devices.

Currently, there are several available IoT controllers / small gateway devices. Examples of these are the Wink Hub[4], Samsung SmartThings Hub[5], Insteon Hub[6] and the Logitech Harmony Hub [7]. These are all closed source controllers. In addition to the closed source controllers there is currently only one strongly active open source controller – openHAB – which can be installed on computers and microcomputers. In our work, openHAB was chosen because it is open source, comprehensive, vendor agnostic, extensible and well documented. More importantly, future work can modify the openHAB source code to ensure the forensic soundness of the state acquisition methods presented in this work to transform it into a true FSAC.

4.2 Forensic State Acquisition Controller (FSAC)

As mentioned, our framework will have its base rooted in what we call a Forensic State Acquisition Controller or FSAC which is based on the openHAB implementation presented in Section 5.1.2. A FSAC should have the same functionality as an IoT controller but with forensics in mind. By definition, for a controller to be regarded as a FSAC, at a minimum, it should embody the following attributes:

- Forensic soundness: A FSAC can only acquire state data from IoT devices and is not allowed to change the state of an IoT device.
- Date & time logging: A FSAC should be capable of accurately logging the dates and times of state changes.
- Secure storage and integrity: A FSAC should embody secure storage of the collected IoT state data, and should also

[3] https://en.wikipedia.org/wiki/Real-time_operating_system

[4] http://www.wink.com/products/wink-hub/

[5] https://www.smartthings.com/

[6] http://www.insteon.com/insteon-hub/

[7] http://www.logitech.com/en-us/product/harmony-hub

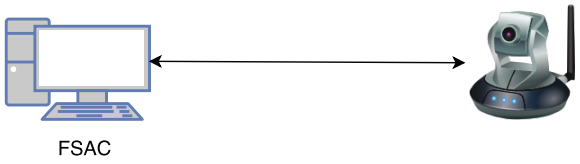

Figure 1: FSAC to Device connection

Figure 2: FSAC to cloud connection

hash the collected states at the time of collection for later validation.

4.3 FSAC to device

Often, IoT device connections are construed as a plain controller to device type connection shown in Figure 1. A simple example would be an IP camera and an application residing on a computer within the same network used to control it. When a camera detects motion it can be registered as a state change. This state change can be reported back to a listening controller where further actions can be taken based on the identified state change.

4.4 FSAC to cloud

Many of today's IoT devices consume cloud services as points of control and data collection. In this mode, IoT device states can be obtained from the cloud data by leveraging APIs which are used to manage IoT devices over the Internet. A prominent example of such APIs is nest's API[8]. The nest temperature controller receives its communication through calls to the cloud (Figure 2). With this access, it becomes possible to employ an FSAC's access to the nest device or the ability to construct individual scripts for direct access to cloud data to log state and state changes of cloud controlled IoT devices.

4.5 FSAC to controller

As mentioned before, controllers connect back to the Internet to allow for simple control via mobile applications or web interfaces. Thus, it is possible to go from a FSAC to a controller as shown in Figure 3. By accessing the controller, the states of multiple devices can be acquired. Obviously, this stands to be a lucrative data collection point in our framework. As companies expand their suite of IoT devices, the need to maintain a central point of control is cultivated. This centralized point of control also promotes a consolidated interface for collecting state data of multiple devices.

[8]According to wikipedia, Nest Labs is a home automation producer of programmable, self-learning, sensor-driven, Wi-Fi-enabled thermostats, smoke detectors, and other security systems.

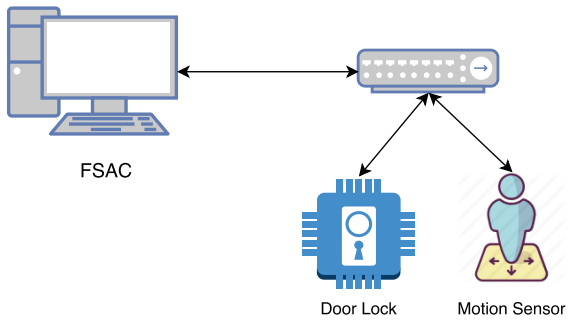

Figure 3: FSAC to controller

5 FSAIOT FRAMEWORK PROOF OF CONCEPT IMPLEMENTATION

It is important to test a proof of concept implementation of the FSAIoT framework. In this section we implemented our proof of concept using the apparatus listed in Table 1. Our framework implementation and proof of concept was a result of three phases:

- Phase I: Setup / installation
  - IoT device selection and configuration, FSAC implementation, and scenario creation.
- Phase II: Log / data acquisition.
- Phase III: Analysis and findings.

5.1 Phase I: Setup / installation

In the following, we first present the selection and configuration of the IoT devices used in our implementation (Section 5.1.1), followed by the FSAC implementation including the connections in Section 5.1.2 and lastly the created laboratory testing scenario in Section 5.1.3.

5.1.1 IoT device selection and configuration. There are many IoT devices that are available for binding given openHAB's extensible architecture (See Table 4) and the devices used in our testing are not the only supported devices. However, to limit the scope of our work, three types of devices were selected. An Insteon IP camera was used as a FSAC to device test, the Insteon Hub was selected as a FSAC to controller test, and the nest thermostat was used as a FSAC to cloud test. Below is a brief synopsis of binding configurations for selected devices:

Insteon IP Camera: The Insteon IP camera is supported by the Insteon Hub. It can be directly controlled via an application that Insteon provides to interface with the hub (controller). openHAB does not officially support control of any IP cameras. We constructed a custom script in order to provide motion detection and video recording. The script employs the ffmpeg[9] package for recording images and video from the IP camera stream. When the script runs, its output is recorded in log files, and the videos are stored accordingly.

Insteon Hub: The Insteon Hub is similar in functionality to the openHAB implementation. It is a controller for multiple IoT devices. It is, however, closed source, and limited to

[9]https://ffmpeg.org/

Table 1: Apparatus

Tool	Description	Utilization
Desktop PC	OPTIPLEX 755 (Intel Core2 vPro)	Prototype
Raspberry Pi 2	900 MHz 32-bit quad-core ARM Cortex-A7 processor	Prototype
Ubuntu 16.04 Server	Linux openHAB 4.4.0-59-generic	OS Used on Desktop PC
Rasbian Jessie Lite	Linux 4.4	OS Used on Raspberry Pi
OpenHAB v1.8	Written in JAVA	Core of Interface to IoT devices
Open HAB Bindings	nest & Insteon Hub	Interface Devices to Open HAB
nest Thermostat v2.0	Home Smart Thermostat	Sample data source
Insteon Hubversion?	IoT Hub	Device controller
Insteon Motion Sensor	IoT Motion Sensor	motions sensor
Insteon Door Sensor	IoT Door sensor	door open/close sensor
Instion IP camera	IoT IP Camera	Video multiaxis rotation
Foscam IP camera	IoT IP Camera	Video multiaxis rotation

Insteon brand IoT devices. In order to acquire state data from devices connected to the hub it is necessary to acquire the device ID. Device IDs is located on actual sensors.
In addition to locating device IDs on the hardware, it is possible to use the insteon-terminal [10]. The terminal allows for enumeration of devices connected to the hub. The terminal requires the username and password that is also available on the bottom of the Insteon Hub.
When openHAB is initialized and correctly configured to support the Insteon Hub it will reach out and attempt to connect to Insteon Hub's modem database. This can be seen in the `openhab.cfg` file. If it cannot find the hardware address in the modem database it will cause error messages. Currently, this appears to be the easiest way to talk to the Insteon Hub and associated hardware devices connected. There does appear to be API access in the works [11]. We applied to acquire an API key[12], however, there has been no response at the time of writing.

nest Thermostat: It is necessary to create both a nest developer account and standard nest user account. Simply creating the developer account will also create a standard user account. Once these accounts are created, logging into the standard user account and adding the nest thermostat is all that is needed to facilitate control via the nest API. A 7 digit entry key is required which can be found via the settings on the thermostat itself. A `nest:client_id`, `nest:client_secret`, and `nest:pin_code` are necessary for communicating with openHAB. The `nest.items` file contains the entries necessary to acquire desired state data.

5.1.2 FSAC implementation and connection. As aforementioned, the core of our setup is openHAB which was installed and configured on a Dell OPTIPLEX 755. While this is an old hardware device, it was sufficient to support Ubuntu Server and openHAB's runtime environment. The use of Ubuntu server limited the number of excess applications being installed while providing necessary support for further customization, as well as the necessary packages to run openHAB. Additionally, Ubuntu allowed for easy installation of openHAB via the APT package management system[13].

[10] https://github.com/pfrommerd/insteon-terminal
[11] http://www.insteon.com/become-an-insteon-developer
[12] http://docs.insteon.apiary.io/#
[13] https://github.com/openhab/openhab/wiki

In parallel to building the openHAB implementation of our FSAC on a desktop, a Raspberry Pi version was tested. The motivation behind testing on a Raspberry Pi was that on an actual case it might be necessary to have a mobile FSAC – should that be the purpose. We contend that more testing is required to validate our initial findings on the Raspberry Pi version.

Note, in its original implementation, openHAB is not a true FSAC implementation. However, our goal was to test the feasibility of acquiring state data from IoT devices. For openHAB to become a true FSAC, future work should modify openHAB so that it is more forensically sound, and that it stores data in a secure and trustworthy manner for future integrity checking. More future work is discussed in Section 9.

Connections. openHAB supports a diverse selection of IoT devices. To talk to IoT devices, openHAB uses bindings to connect the IoT devices to the run time backend for which the `openhab.cfg` file needs to be edited. This file contains a list of all the devices currently supported by openHAB. Each item has a section called a binding. Within the binding section, the necessary connection information must be configured. If a particular device is currently not supported, it is also possible to code bindings.

In addition to the `openhab.cfg` file, each device needs to have a `*.item` file located in the `items` folder on the controller. The `*.items` file contains the specific states to monitor.

To apply the FSAIoT framework discussed in Section 4, we tested the three different modes of obtaining forensically relevant IoT state data:

- FSAC to device: FSAC communicates directly to the device.
- FSAC to controller: FSAC communicates to an IoT Hub such as the Insteon Hub.
- FSAC to cloud: FSAC obtains data via a cloud API.

Although much of the data aggregation is performed through openHAB, some devices cannot be interfaced with openHAB and must be connected using another approach such as script writing for interfacing with the IoT devices and acquiring their state data.

5.1.3 Scenario Creation. The experiment took place in our lab (Figure 4); which consists of one large room and two small rooms separated from the large room by doors. One of the small rooms contains a server. The lab was set up with three IoT devices – door sensor, motion sensor and an IP camera. The door sensor was placed on the door between the large room and the hallway to record entry

and exit to and from the lab. The motion sensor was placed in the server room to detect motion. This was to detect movement to and from the server room. The IP camera was located inside the server room to monitor activity.

The Dell OPTIPLEX 755 running Ubuntu Server was set up to monitor and record data from all of these devices using our openHAB FSAC implementation. At some point during the experiment, a fictitious malicious individual made an attempt to access the server. In addition to the hardware used in this scenario, a nest thermostat was employed to provide an example of the FSAC to cloud mode presented in the FSAIoT framework (Section 4).

The researchers additionally recorded the events along with the date and time each event occurred (e.g., entering the room) to cross-validate our implementation findings. Lastly, the data captured by our FSAC implementation was analyzed to construct a timeline (Figure 5) of events and attempted to determine what had occurred while comparing against the researcher notes.

A note about the scenario. Good security practices assume that a large scale organization should be logging entry and exit, video, audio and any other useful security data points. Where this scenario is more likely to take place could be in a small business / home.

5.2 Phase II: Log / data acquisition

The openHAB FSAC implementation provided several methods of data recording. Databases can be configured locally or remotely, data can be pushed to a cloud service, or logs can be kept locally. In the implemented FSAC prototype we decided to use log files stored in `/var/log/openhab`. The log files that openHAB keeps locally contain the necessary raw data to show device states with their respective time stamps.

Dropbox was also used to save log files and associated video recorded using a custom script. This allowed for long term storage and off site analysis. It is possible to set the timing for how often this push occurs. If set to a higher frequency, analysis can occur near real time. openHAB provides a method to select other logging capabilities should one desire an alternative.

5.3 Phase III: Analysis & findings

Within the log files are the states and the changes that occur, as shown in Table 2. We selected the states to monitor in the *.item file. With regards to the nest, there are states that would be considered forensically useful. Examples consist of the timezone the device is in, the location of the device by room name (this can be hard to verify without gaining physical access to the nest as it is a custom name that nest suggests be the room name). These states, and other forensically relevant states from the nest are exemplified in Table 3.

We note something about accessing data from the Insteon Hub. The Insteon system is largely event driven with a few exceptions including the thermostat and dimmer module. This means that Insteon Hub's sensors cannot have their initial states pulled, and it is not until the sensor registers an event that openHAB is updated with the state of the sensor[14]. As a result, the web interface shows unknown as the current state. Once the sensor experiences a state change, it updates showing the current sensor state. Specifically, if an investigator wishes to know the previous state of a binary

[14]https://github.com/openhab/openhab/wiki/Insteon-PLM-Binding

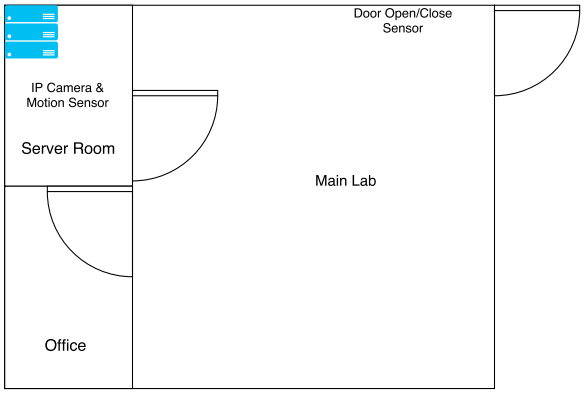

Figure 4: Lab scenario layout showing IoT device locations

sensor, it is safe to assume it is the opposite of whatever the most current state is. As an example, once openHAB is initialized, the door sensor reads unknown. If we assert that the actual state of the door is closed and the door is opened, the web interface and logs are updated with door open. Therefore, the door's initial state was closed. This has been tested and verified in our analysis.

As mentioned in Section 5.1.1, the IP camera is a stand alone device in our scenario. The custom script outputs an `*.mp4` file as well as logs showing when the camera saw motion. With both of these files it becomes possible to have both visual and text based confirmation of who caused the motion and when it occurred.

After running our scenario, a set of states were collected. An excerpt of the log entries of interest are shown in Table 5. We will use these entries to paint a picture of the events that took place.

In Figure 4, the facility and its layout are illustrated. This layout could have been obtained perhaps by public access records or during an initial set up or reconnaissance of the facility. Knowing what IoT devices are available and where they are located is important to help map out events that occurred.

Recall, the scenario described in Section 5.1.3. The intent of placing our FSAC implementation on-site was to monitor the current IoT devices and obtain state changes that would help convict a possible intrusion and attempted access to private data. Examining the logs generated in Table 2, one can clearly see when doors open / close, motion in a server room occurs, and when this motion is recorded by the camera. With this evidence and if an investigator had access to the logs on a victim's server, it would be possible to describe with detail the events that took place.

In Table 2, FSAC to controller and to device can be seen effectively collecting states. The door and motion sensor fall under the FSAC to controller portion of the framework and the camera falls under FSAC to device. The additional testing of the framework not seen in our scenario is the FSAC to cloud state acquisition. However, the nest was configured and data was collected as a proof of concept. Table 3 shows the states that were collected.

6 DISCUSSION

Our findings show that indeed IoT device state and or the state change could be of forensic value. To substantiate our claim, queries

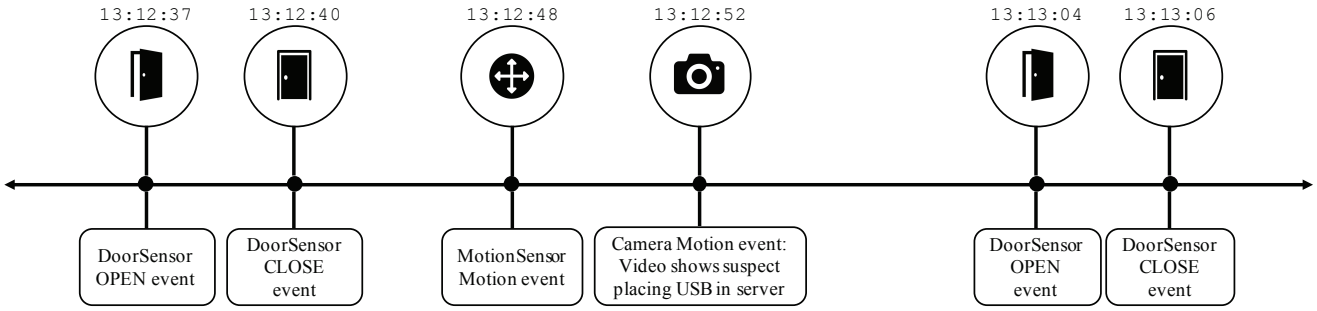

Figure 5: Timeline of events

Table 2: Scenario log excerpt

Date/Time	Description	State/File name
2017-01-10 13:12:37	doorSensor state updated to	OPEN
2017-01-10 13:12:40	doorSensor state updated to	CLOSE
2017-01-10 13:12:48	motionSensor state updated to	OPEN(motion)
10-01-2017 13:12:52	Camera motion detected. Recording:	20170110131252.mp4
2017-01-10 13:13:04	doorSensor state updated to	OPEN
2017-01-10 13:13:06	doorSensor state updated to	CLOSED

Table 3: nest state captures

Date/Time	Description	Status
2016-11-27 00:00:16	den_room_target_temperature_f state updated to	72
2016-11-27 00:00:16	den_room_ambient_temperature_f state updated to	71
2016-11-27 00:00:16	home_away_time_zone state updated to	America/New_York
2016-11-27 00:01:16	home_away_structure_id state updated to	mP0UQIuD9x9TxDPETEqF1AGiXdB-WMSS7ox4C3QE32U8jSvOgWdqg
2016-11-27 00:01:16	home_away_device_id state updated to	DGYsZuvftgdLwhTaxwPKx50qHK5YW7p
2016-11-27 00:01:16	den_room_last_connection state updated to	20161127T00:01:11
2016-11-27 00:01:16	home_away state updated to	home

were performed to show both the current trend in IoT devices and what sort of access might be available for these devices.

With that said, our method, provides a general solution towards device state acquisition it does not physically acquire the memory of the devices themselves. If we switch focus to the future of our proposed approach, it becomes important to understand the future of APIs in the IoT domain.

A forensically positive trend among many of the device controllers such as the Wink Hub, Nest platform and Insteon Hub, is the desire for these companies to offer API support. The goal being to provide developers the ability to integrate device status and control methods into their applications. The three devices listed above each have their own developer sites as well as APIs that are available to those that apply for access. Once an API key is granted, API calls can be made via whatever programming languages are supported by the manufacturer.

Table 4 provides a list of some of the most popular IoT devices today along with their potential compatibility with our FSAC implementation – openHAB. openHAB compatibility is characterized by a device having either an open API that we can leverage or official support from openHAB through a binding. They are ordered by relevance of category, beginning with security devices. 9 out of 23 security devices have some sort of compatibility with openHAB, whether it be through an API or through openHAB bindings. This number is quite low and represents an area that we will need more research by either constructing openHAB bindings or custom tools to obtain state data. As for home automation devices, 16 out of 24 devices seem to be openHAB compatible. It is worth noting that a few of the home automation and security devices actually fit into both categories, such as the Insteon Hub.

Table 5 is a continuation of Table 4, again organized by category. There are several devices on the market for tracking. These may include asset, fleet, or human tracking, and are most applicable in an industrial or commercial environment. Half of these devices would be compatible with our FSAC implementation at this time. The next two blocks are appliances and entertainment devices. Only one appliance is supported whereas all but one of the entertainment devices are supported. The last block, miscellaneous, contains devices that do not fit into any of the preceding categories. In general, these devices do not align with FSAC objectives and happen to have low amount of support. There are, however, four Wi-Fi router devices that may be forensically interesting, but none of them are currently openHAB compatible. This may be another area worth dedicating research efforts for exploring future support.

In this work, nest's API was leveraged. However, only a few of the API calls were used. There are many more API calls that could be employed. To show the rest of states that may be pulled beyond what was used in our scenario, we share below nest's API states:

Thermostat: `Device_id, locale, software_version,name, name_long, last_connect, is_online, is_using_emergency _heat, target_temp_f, hvac_mode, where_id, is_locked, where_name, previous|-havac_mode`

Smoke/CO alarm: `device_id, name, structure_id, name_long, last_connection, smoke_alarm_state, where_name, software_version, last_manual_test_time, co_alarm_state, is_online, where_id`

Cameras: `device_id, software_version, structure_id, where_id, where_name, public_share_url, name_long, is_online, is_streaming, app_url, is_audio_input_enabled, is_public_share_enabled, is_video_history_enabled, name, last_is_online_change, snapshot_url, last_event`

While the nest API only supports three major devices currently, the Wink Hub and Insteon Hub support even more hardware, and while Insteon's API is in its early stages of development, Wink Hub's API is fully functional allowing for similar API calls to access connected hardware.

As the reader can see, the presented states go beyond what was presented in the simple scenario created. Two of the devices listed; the Wink Hub and Insteon Hub, both are centralized controllers. This will allow future FSAC implementations the ability to acquire state data from a multitude of possible IoT devices.

7 LIMITATIONS

The biggest limitation both in IoT forensics and with the FSAC prototype is accessing historical and deleted data. Currently, the FSAC proof of concept can pull the state of the nest thermostat. However, the Insteon door sensor and motion sensor will only allow the acquisition of the change of state as mentioned in Section 5.3. The second challenge lies in the need for physical access. This is a common challenge with regards to digital forensics and security. Another limitation lies in connecting to the different IoT devices. In particular, the different wireless connection methods. Currently the more common methods used are wireless (802.11a,b,g,n,ac), Bluetooth, Zigbee, and Zwave. In order to acquire data via Zigbee or Zwave it is necessary to have hardware that supports these communication technologies. While this hardware is not common place on a traditional computer or the Raspberry Pi, modules that support these wireless standards are available.

Our work has limitations in that we did not explore the forensic soundness of the implemented IoT acquisition controller. We deemed that as future work. We also did not examine the robustness of the implemented system by conducting test-retest reliability experiments. Our overall goal was to examine the feasibility of the proposed framework. We do not discount the importance of robustly testing our proposed framework and its generlizability but we argue that this work is the initial step towards studying the utility of the FSAIoT framework.

8 CONCLUSION

In this work we argued that there are many challenges related to IoT device forensics. We presented the FSAIoT framework, and illustrated its feasibility by implementing a proof of concept FSAC using OpenHAB as well as self-created scripts to validate the feasibility of the framework. We showed that we are able to reliably collect state data from IoT devices using the three different modes: controller to device, controller to cloud and controller to controller. Even considering the limitations, we contend that our proposed framework and FSAC proof of concept are essential steps in evolving a general methodology for obtaining valuable forensic evidence from a diverse and thriving climate of IoT devices.

9 FUTURE WORK

Although this work shows promising results in our ability to acquire state data from IoT devices in a forensic investigation, we present research opportunities for expanding our work.

9.1 Forensic soundness

Many of the devices, such as dimmer modules or the thermostats can have their states modified through the traditional openHAB environment. This can be detrimental to the forensic process as it would contaminating evidence. To protect against this, a full review of the openHAB codebase should be performed to establish which code blocks result in changing the state of particular devices, and in what way. In some cases it may be necessary for openHAB to write to a device in order to establish a connection or perform other functions, but from a forensic perspective, it must be known exactly what changes during the process. Other cases which involve human intervention (adjusting the room temperature or dimming the lights) should be disabled in the source code. This opens the door for research in attempting to find general methods for an IoT device state hardware / software writeblocking implementation.

9.2 User experience

It is critical for production purposes to implement a FSAC that is more user friendly with a forensic-centric user interface. Currently, in order to make use of all the features, a user needs to interact with both the stock openHAB web interface as well as a considerable use of the terminal. Terminal use is necessary for the setup process to edit the various configuration files as well as for setting up the IP camera recording. It would be useful to centralize all of these tasks in a single web application where an investigator may perform 100% of the setup and analysis without ever having to use the terminal. Furthermore, adding a visualization engine that enables investigators to create time lines would be useful. This opens the avenue for visualization research for IoT state data.

9.3 IoT network device fingerprinting

In order to make our proposed approach friendly to investigators, it is worth exploring practical methods of automatically fingerprinting and triaging IoT devices on a network under investigation. This can help speed up the identification process in the digital forensic process.

9.4 Network packet and reconstruction

It would be worthwhile to implement and research network packet data between a FSAC and IoT devices so that more control may be gained by technically trained network forensic examiners and researchers for data that may not be available through a FSAC user

interfaces. This also opens the door to network protocol reverse engineering researchers for methods of acquiring state data from IoT devices.

9.5 Deleted data

Another important area of research is to clearly identify IoT devices that do indeed store data and find mechanisms for the physical acquisition of their memory for further analysis.

REFERENCES

[1] Tamas Abraham and Olivier de Vel. 2002. Investigative profiling with computer forensic log data and association rules. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on*. IEEE, 11–18.

[2] Ibrahim Baggili, Jeff Oduro, Kyle Anthony, Frank Breitinger, and Glenn McGee. 2015. Watch what you wear: preliminary forensic analysis of smart watches. In *Availability, Reliability and Security (ARES), 2015 10th International Conference on*. IEEE, 303–311.

[3] Konstantia Barmpatsalou, Dimitrios Damopoulos, Georgios Kambourakis, and Vasilios Katos. 2013. A critical review of 7 years of Mobile Device Forensics. *Digital Investigation* 10, 4 (2013), 323–349.

[4] Simson L Garfinkel. 2010. Digital forensics research: The next 10 years. *digital investigation* 7 (2010), S64–S73.

[5] Pavel Gladyshev and Ahmed Patel. 2004. Finite state machine approach to digital event reconstruction. *Digital Investigation* 1, 2 (2004), 130–149.

[6] Sharon P Hall and Eric Anderson. 2009. Operating systems for mobile computing. *Journal of Computing Sciences in Colleges* 25, 2 (2009), 64–71.

[7] Vikram S Harichandran, Frank Breitinger, Ibrahim Baggili, and Andrew Marrington. 2016. A cyber forensics needs analysis survey: Revisiting the domain's needs a decade later. *Computers & Security* 57 (2016), 1–13.

[8] Andrew Marrington, Ibrahim Baggili, George Mohay, and Andrew Clark. 2011. CAT Detect (Computer Activity Timeline Detection): A tool for detecting inconsistency in computer activity timelines. *digital investigation* 8 (2011), S52–S61.

[9] Edewede Oriwoh, David Jazani, Gregory Epiphaniou, and Paul Sant. 2013. Internet of Things Forensics: Challenges and approaches. In *Collaborative Computing: Networking, Applications and Worksharing (Collaboratecom), 2013 9th International Conference Conference on*. IEEE, 608–615.

[10] Edewede Oriwoh and Paul Sant. 2013. The forensics edge management system: A concept and design. In *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*. IEEE, 544–550.

[11] Joseph Ricci, Ibrahim Baggili, and Frank Breitinger. 2016. Watch What You Wear: Smartwatches and. *Managing Security Issues and the Hidden Dangers of Wearable Technologies* (2016), 47.

[12] Ting Sang. 2013. A log based approach to make digital forensics easier on cloud computing. In *Intelligent System Design and Engineering Applications (ISDEA), 2013 Third International Conference on*. IEEE, 91–94.

[13] Shams Zawoad and Ragib Hasan. 2015. FAIoT: Towards Building a Forensics Aware Eco System for the Internet of Things. In *Services Computing (SCC), 2015 IEEE International Conference on*. IEEE, 279–284.

[14] Ben Zhang, Nitesh Mor, John Kolb, Douglas S Chan, Ken Lutz, Eric Allman, John Wawrzynek, Edward A Lee, and John Kubiatowicz. 2015. The Cloud is Not Enough: Saving IoT from the Cloud.. In *HotCloud*.

A IOT APIS AND OPENHAB COMPATIBILITY

Table 4: IoT APIs and openHAB compatibility

Manufacturer	Device	Description	Open API	openHAB	Coverage
Category: Security					
Schlage	Schlage Door Locks	Smart home locks	No	No	No
Ring	Ring: Video doorbell	Doorbell	No	Yes	Yes
Scout	Scout Alarm	No	No	No	
Garageio	Garageio	Connected garage door	No	No	No
Nest	Dropcam	Connected camera - Now Nest Cam	Unofficial	No	Yes
August	Smart Keypad	Smart keypad	Yes	No	Yes
August	Doorbell Cam	Doorbell cam	Yes	No	Yes
Canary	Canary system	No	No	No	
Chamberlain	MyQ	Garage door opener	Yes	Yes	Yes
Netatmo	Netatmo Welcome	Indoor security camera	Yes	Yes	Yes
Piper	Piper	Home security and a home automation hub	No	No	No
CUJO	CUJO	Smart firewall for the smart home	No	No	No
Logitech	Logi Circle	Portable Wi-Fi video camera	Yes	Yes	Yes
Lockitron	Bolt	Smart Lock	Yes	No	Yes
Sentri	Sentri	Home monitoring and automation	No	No	No
Tripper	Tripper	Window and door sensor	No	Unofficial	Yes
Arlo	Arlo	Smart home security camera system	No	No	No
BeON	BeON	Home security	No	No	No
Cocoon	Cocoon	Home security	No	No	No
Point	Point	Home security	No	No	No
Homeboy	Homeboy	Home security	No	No	No
Kibbi	Kibbi	Home security	No	No	No
Leeo	Leeo	Nightlight with home safety functionality	No	No	No
Category: Home Automation					
Wink	WinkHub	Smarthome hub that connects together many other IoT products	Yes	No	Yes
Amazon	Echo	Smarthome hub	Yes	Yes	Yes
Nest	Nest	Internet-connected thermostat, smoke and carbon monoxide detectors, and cameras	Yes	Yes	Yes
Google	Google Home	smarthome hub	Yes	Unofficial	Yes
Insteon	Insteon Hub	Smarthome hub - connects several devices together for home automation	No	Yes	Yes
Honeywell	Smart House Products	thermostats, GPS asset tracking, locks, lighting, video surveillance and more	Yes	No	Yes
Belkin	WeMo	Home IoT devices that includes smart switches, cameras, lights, an air purifier, heater, slow cooker, humidifier and more	No	Yes	Yes
Samsung	SmartThings	smart outlets, hubs, motion sensors, multipurpose sensors, arrival sensors, water leak sensors and more	Yes	No	Yes
GE	Link	Lighting	No	Yes	Yes
OSRAM	Lightify	Lighting	Yes	Yes	Yes
Philips	Hue	Lighting	Yes	Yes	Yes
Awair	Awair	Air-quality sensor	No	No	No
Elgato	Eve	Monitor indoor air, outdoor weather, energy consumption	Yes	Yes	Yes
Eversense	Eversense	Thermostat: senses where your smartphone is and adjusts the temperature in each individual room accordingly	No	No	No
Keen	Home Smart Vent	Opens and closes the vents in each room depending on the temperature	No	No	No
Bosch	Smart Home	Sensors that go into other IoT devices as well as some smart home appliances	Yes	No	Yes
Logitech	Pop	Smart button that allows you to control connected devices	Yes	Yes	Yes
Ivee	Ivee Voice	Voice control for the home	No	No	No
Logitech	Logitech	Harmony Elite Home Controller	Yes	Yes	Yes
Switchmate	Switchmate	Smart lighting made simple	No	No	No
NEEO	NEEO	Smart home remote	No	Yes	Yes
Quirky	Outlink	Smart outlet	No	No	No
Onecue	Onecue Gesture	Control for your home	No	No	No
Tapt	Tapt	Smart light switch	No	No	No
Brio	Brio	Smartest power outlet	No	No	No

Table 5: IoT API's and openHAB compatibility (cont)

Manufacturer	Device	Description	Open API	openHAB	Coverage
Category: Tracking					
Awarepoint	Awarepoint	Track the locations of employees, assets, customers, patients and more in real time	On request	No	Yes
ATrack	ATrack trackers	GPS tracking for monitoring assets and vehicles	No	No	No
DorsaVi	ViSafe	Track how employees are moving	No	No	No
Impinj	RFID sensors	Tag chips, gateways, readers, antennas and software for retailers, health care and other markets	Yes	No	Yes
Samsara	Sensors	IoT sensors for fleet telematics, energy monitoring, cold chain monitoring, asset monitoring and other purposes	Yes	No	Yes
Xerafy	RFID sensors	RFID tags and other technology for asset tracking	No	No	No
Category: Appliances					
GE	GE Connected Appliances	wall ovens, ranges, refrigerators, dishwashers, washers and dryers, water heaters and air conditioners	No	No	No
Whirlpool	Smart Appliances	No	No	No	
June	Intelligent Oven	countertop oven	No	No	No
Cinder	Cinder	Countertop grill	No	No	No
LG	SmartThinQ	Kitchen (ranges and refrigerators), living (washers, dryers, robotic vacuums and air conditioners) and safety (robot vacuum doubles as a safety monitor with a video feed)	No	Yes	Yes
Nespresso	Prodigio	Coffee maker	No	No	No
Maid	Maid	Microwave Oven	No	No	No
Category: Entertainment					
Sonos	Sonos	Smart speaker system	Yes	Yes	Yes
Apple	Apple TV	Home entertainment	Yes	No	Yes
Google	Chromecast	Home entertainment	Yes	Yes	Yes
Roku	Roku 3	Streaming media player	Yes	Yes	Yes
Ray	Super Remote	Universal remote	No	No	No
Category: Miscellaneous					
Amazon	Amazon Dash Buttons	Buttons to order additional supplies of commonly used household products directly from Amazon.com	Yes	Unofficial	Yes
Google	Google Wifi	Boost home WiFi signal dead zones.	No	No	No
Eero	Eero	WiFi system	No	No	No
Starry	Starry Station	WiFi wireless router	No	No	No
Google	OnHub	Connected home router	No	No	No
Neurio	Neurio	Track energy usage	Yes	No	Yes
Roost	Roost	Battery tells you when it needs to be changed	No	No	No
Toymail	Talkies	Talkies combines a lovable friend with a built in smartphone for communication	No	No	No
Mimo	Mimo	Smart baby monitoring	No	No	No
NUZii	NUZii	The world's first all-in-one smart life device	No	No	No
Evermind	Evermind	Connected home care	No	No	No
AdhereTech	AdhereTech	Smart, wireless pill bottles	No	No	No