Digital Investigation 22 (2017) S3–S14

Contents lists available at ScienceDirect

Digital Investigation

journal homepage: www.elsevier.com/locate/diin

DFRWS 2017 USA — Proceedings of the Seventeenth Annual DFRWS USA

DROP (DRone Open source Parser) your drone: Forensic analysis of the DJI Phantom III

Devon R. Clark*, Christopher Meffert, Ibrahim Baggili, Frank Breitinger

Cyber Forensics Research and Education Group (UNHcFREG), Tagliatela College of Engineering, ECECS, University of New Haven, 300 Boston Post Rd., West Haven, CT 06516, USA

Keywords:
Computer forensics
Cyber forensics
Digital forensics
IoT forensics
Embedded systems forensics
UAV forensics
Drone forensics
Mobile forensics
Drone
UAV
DJI
Phantom III
DJI Phantom III
Open source DRone Parser
DAT file structure
TXT file structure

A B S T R A C T

The DJI Phantom III drone has already been used for malicious activities (to drop bombs, remote surveillance and plane watching) in 2016 and 2017. At the time of writing, DJI was the drone manufacturer with the largest market share. Our work presents the primary thorough forensic analysis of the DJI Phantom III drone, and the primary account for proprietary file structures stored by the examined drone. It also presents the forensically sound open source tool DRone Open source Parser (DROP) that parses proprietary DAT files extracted from the drone's nonvolatile internal storage. These DAT files are encrypted and encoded. The work also shares preliminary findings on TXT files, which are also proprietary, encrypted, encoded, files found on the mobile device controlling the drone. These files provided a slew of data such as GPS locations, battery, flight time, etc. By extracting data from the controlling mobile device, and the drone, we were able to correlate data and link the user to a specific device based on extracted metadata. Furthermore, results showed that the best mechanism to forensically acquire data from the tested drone is to manually extract the SD card by disassembling the drone. Our findings illustrated that the drone should not be turned on as turning it on changes data on the drone by creating a new DAT file, but may also delete stored data if the drone's internal storage is full.

Introduction

Unmanned Aerial Vehicles (UAV)s, also known as drones, have become increasingly popular in recent years, largely due to the rapid increase in affordability for the average consumer. They are no longer expensive machines (less than $1000) and can often be controlled with ease, e.g., utilizing a smart phone application for direct control or feedback.

Dà-Jiāng Innovations Science and Technology (DJI), a Chinese company, acquired success in the consumer drone market. DJI is one of the most prominent drone manufacturers today. A dronelife.com article reports a revenue jump from $4.2 million in 2011 to $130 million in 2013. An estimated 533,400 drones have been sold between their initial launch and the end of 2014 (Amato, 2015). DJI produces two primary aircraft models, the Phantom series and Inspire series. This work focused on the forensic analysis of the Phantom III Standard, launched in April of 2015.

With this increase in popularity, the need for laws regarding the use of drones has also increased. The Federal Aviation Administration (FAA) has reported 583 drone incidents from August 2015 to January 2016. These incidents usually involved curious or mischievous pilots flying their drones into restricted airspace. The Federal Bureau of Investigation (FBI) has been asked to investigate rogue drones, with the most challenging task being the tracking of pilot fault, according to Brandom (2016). To keep up with technology, the FAA has been working to develop laws to constrain the use of recreational UAVs. These laws mostly limit the maximum altitude and impose no-fly zones for airports and other areas such as recreational sporting events with more than 30,000 people.

Drones have also had an impact on terrorism activities. For example, the Islamic State of Iraq and the Levant (ISIL) has been using drones for surveillance purposes for some time now as Schmidt (2016) points out. Furthermore, there have been three recorded incidents (one fatal) of ISIL rigging the drones with

* Corresponding author.
E-mail addresses: dclar4@unh.newhaven.edu (D.R. Clark), cmeff1@unh.newhaven.edu (C. Meffert), IBaggili@newhaven.edu (I. Baggili), FBreitinger@newhaven.edu (F. Breitinger).
URL: http://www.unhcfreg.com/, http://www.Baggili.com/, http://www.FBreitinger.de/

http://dx.doi.org/10.1016/j.diin.2017.06.013

explosives. The terrorist group is not using military grade aircrafts, but rather they are resorting to commercially available drones which include the DJI Phantom series, the very same series investigated in this work. Due to the growth in popularity of commercial UAVs along with the spike in criminal activity involving them, there is a dire need for the development of sound UAV forensic techniques. We posit that this need will continue to increase as UAVs become more accessible and employed in other criminal activities.

To the best of our knowledge, we present the first comprehensive work on the forensic analysis of the DJI Phantom III Standard drone. Our contribution in this work is as follows:

- We present a set of procedures that examiners may learn from during the process of investigating a case that involves the DJI Phantom III Standard drone.
- We present the first openly accessible account for the binary (FLYXXX.dat) file structure – the flight recording file created by the drone and stored on its SD card.
- We compile our findings into the only available open source python tool called DRone Open source Parser (DROP) capable of parsing the proprietary DAT file format in a forensically sound manner. This tool is available for download at https://github.com/unhcfreg/DROP.
- We provide the first account, tool, and method for being able to correlate data extracted from the drone's nonvolatile internal storage and the mobile device used to control it.

We first start the paper by stating the scope of our research (Section Research scope). We then review previous work related to UAV forensics in Section Related work. This is followed by the methodology and process used to conduct the forensic analysis of the Phantom III in Section Methodology. In Section Data analysis, we present the analysis of data acquired from our methodology and illustrate the drone's proprietary file structures, and a tool used to parse one of them in Section Tool creation: DRone Open source Parser (DROP). Next, we discuss the results obtained from our analysis in Section Testing and findings. Section Limitations discusses the limitations associated with our work and DROP. Finally, our concluding thoughts are shared in Section Conclusions & future work. We assert that one limitation of our work in this paper is that it encompasses just the DJI Phantom III Standard model. Further efforts to extend the research to additional DJI models and other manufacturers may be addressed in future work.

Research scope

It is important to note that the scope of this work is limited to the DJI Phantom III standard drone. While it would be ideal to test a multitude of drones, the work is rather tedious and requires plenty of reverse engineering. It is also of note that the idea of implementing a generalized solution that would be capable the forensic analysis of all the drones available on the consumer market is a strenuous exercise. Each drone is different in terms of its operating system, proprietary data storage, and control protocols making the task for forensic examiners difficult. While we dive deep into the DJI Phantom III, we argue that this is a good start given that DJI currently holds the highest market share in the Drone market, and the fact that their Drones have already been used by terrorism groups such as ISIL.

Related work

The following sections review several developing areas in digital forensics relevant to drone forensics.

Drone forensics/security

In the process of researching this topic, modest peer reviewed related literature was identified. Much of the literature at the time of writing focused on the security of drones.

Samland et al. (2012) outlined a detailed analysis using the Computer Emergency Response Team (CERT) taxonomy. The CERT taxonomy is a common language by which security related incidents are discussed by experts (Kiltz et al., 2007). By leveraging this taxonomy, Samland et al. (2012) were able to identify potential vulnerabilities. They conducted studies based on three different scenarios: high-jacking the drone; malicious code delivery to the drone; and using the drone's GPS system to track the end user. In each scenario, they shared the particular systems and level of compromise to those systems needed to execute each scenario. Furthermore, they identified protection requirements to help prevent their discovered attacks.

In other work, a presentation by Kovar (2015) discussed the forensic analysis of the DJI Phantom II, the precursor to the Phantom III. He noted the new wave of illegal activity involving drones which included drug and weapon delivery, invasion of privacy, flight in controlled airspace, and flight into bystanders. In his analysis, he was able to recover a plethora of data to trace the aircraft back to the owner. This included GPS and other EXIF data from pictures, launch point, DJI account information, and the owner's name. The work also showed that practitioners can establish a Secure Shell (SSH) connection to the drone and dump the root file system and hijack it in real-time using a tool called Skyjack by Kamkar (2013). Although the work was similar to ours 1) The work did not focus on the Phantom III and 2) A validated forensic methodology and tool was not the focus of the work.

Another presentation by Luo (2016) was shared at DEFCON. This presentation portrayed a detailed security analysis of the DJI Phantom III – the drone used in our work. In the work, Aaron Lou broke down the security analysis into several parts which included: breaking the Software Development Kit (SDK) authentication, firmware analysis, radio signal analysis, and GPS analysis. In each of these, he briefly identified possible compromises as well as solutions. This work focused on the security analysis of the DJI Phantom III and did not focus on the forensics of the drone.

In other recent work, Horsman (2016) discussed a preliminary analysis of forensic challenges for UAVs. The paper presented a forensic analysis of the Parrot Bebop UAV, another popular drone consumer model. The results demonstrated the ability to recover flight data from both the drone and the controller handset. Problems still existed however for establishing ownership of the drone. Similar to David Kovar's analysis of the Phantom II, the researcher was able to connect remotely via Telnet and acquire an image of the root file system.

Thiobane (2015) recognizes that the increased use of drones by private citizens escalates the challenges faced by digital forensic investigators. Among other findings, they identified that drones are targeted by criminals for their payload value, data breach, and cyber-attack capabilities. Several recommendations were made which include creating an academic course to keep Information Technology (IT) and forensic examiners up-to-date on developments in UAV usage. They also proposed that drone manufacturers develop built-in technology to locate hijacked, stolen, and flyaway drones.

As shown from the literature review on drone forensics, peer reviewed work on the forensic analysis of the DJI Phantom Drone III had not been published at the time of writing. Our work improves the state of the art by filling that research gap by both presenting our findings, as well as enabling investigators to forensically analyze proprietary files to recover a slew of digital evidence from the tested drone.

Legal challenges of drones

A major challenge that follows civilian drone usage is privacy. The concerns of the public are a result of cases in which the drone is used to take photo or video of a person unaware of the drone. The case for regulating law enforcement drone usage will not be nearly as much of a challenge as the use of drones by civilians. There is a need to strike a balance between what is acceptable for private use and what is not. Currently, there are laws being created both by state and federal agencies. Many of these laws revolve around the first and fourth amendments. In particular, the need to know the location of the drone when it is recording is critical for privacy (Kaminski, 2013).

Work conducted by Carver (2014) dives into the details of laws against drones at the state level. Some states are simply banning them outright while others are more focused on what is allowed and what is not. One example being laws against drones carrying weapons. The state of Connecticut has proposed a law to restrict/prohibit putting guns on drones after a man posted a video of his drone flying with a gun and shooting according to (Rondinone, 2015).

Ravich (2015) built a detailed analysis of the challenges faced by the courts in relation to drones. Drones are not simply flying machines, but contain information. With so much data capable of being collected with these devices it becomes critical to identify what can or cannot be used in a court of law. A case is made that data acquired via a drone should be as valid as data collected via any other aerial observation tool, as well as any digital item (mobile phones, smart watches, etc).

Clarke and Moses (2014) pointed out that in both the United States and Australia, there are no recorded collisions with civilian drones and air transportation. There has, however, been cases of pilots identifying these types of drones in airspace that makes up airports such as the Perth airport. As of April 17 2016 there has been a recorded incident of a drone striking a British Airways aircraft landing in Heathrow airport (Wild et al., 2016).

In terms of military drones, in 2005 there were two collisions involving drones and aircrafts while in Iraq. According to Zenko (2012), there has been 79 crashes of drones as of 2010 in the United States Air Force.

Phantom III related files and software tools

Previous research found that the DJI Phantom III has two proprietary file formats: binary files (located on the drone's nonvolatile internal storage with the ".dat" file extension) and additional binary files (located on the Android device with the ".txt" file extension used to control the drones). From this point forward, we will refer to the binary ".dat" files as "DAT files" and ".txt" files are referred to as "TXT files." Both of these files are encoded and encrypted (therefore the TXT file is not in the standard ASCII format). At the time of writing, some tools were available for parsing these files. We would like to point out that these tools were not forensic tools, but were created by hackers and hobbyists.

DatCon,[1] which is written in Java was the only tool found capable of parsing DAT files. It is unclear who the inventors/authors are as they remained anonymous. Three tools were identified for TXT files. First, *dji-log-parser* by Franklin (2016), is an open source tool that operates locally in a browser. It was only able to parse data from an older TXT file format, however, it failed with the newer format. Secondly, *CsvView*[2] was found which is closed source and had the same problem. Lastly, there was *Healthy Drones*[3] which is a free online service to convert TXT files to CSV files. Unfortunately, the authors did not release any information on how they were able to extract the data. Additionally, an examiner would have to upload/share evidence with an unknown service provider which is clearly a drawback. – On p. 3, the paper states – It is also worth noting that our research focused on the analysis of the mobile device connecting to the drone with the hope that data between the drone and the controller may be correlated. This sentence gave the impression that the research did not focus on the analysis of data onboard the Drone, which seems to not be the case. This sentence should be removed or clarified.

It is also worth noting that our research included the analysis of the mobile device connecting to the drone with the hope that data between the drone and the controller may be correlated.

Methodology

The apparatus used throughout this research is presented in Table 1. Our work embodied the methodology described in the following step-wise procedure:

Factory reset: To ensure no external variables affected our results, a factory reset and formatting of all devices and cards was first employed (Section Factory reset).

Scenerio creation: The drone was powered on, flown in two separate geographical locations, and then powered off (Section Scenario creation).

Data acquisition: The process of acquiring data was broken into three parts: drone, controller, and phone/tablet (Section Data acquisition).

Data analysis: The acquired data was analyzed. Particularly, two files of interest containing flight data were acquired. An in-depth analysis of these files and their structures was attempted (Section Data analysis).

Tool creation: Once the data was analyzed, and the proprietary file structures were understood, a tool was created to enable examiners to parse evidentiary files (Section Tool creation: DRone Open source Parser (DROP)).

Testing: Finally, a number of tests were conducted to validate our constructed tool and findings. Tests were also carried out with regards to the internal SD card in the drone (Section Testing and findings).

Factory reset

The first step in the process was to factory reset both the drone and the Nexus 7 tablet. The tablet was factory reset by booting into the device recovery menu and selecting Factory Reset. The tablet was then updated to the latest operating system – Android 6.0.1. Next, the drone was factory reset using a secondary Android device with DJI GO installed. DJI GO is the Android application developed by DJI which serves as a command and control dashboard for the Phantom III. This application allowed the user to control the drone, in part, from a mobile device or tablet. It also allowed the user to clear both the flight route data and the video cache from the nonvolatile internal storage device on the drone. As an extra precaution, the 64 GB external SD card located on the camera rig was deleted and formatted to a FAT32 file system.

Scenario creation

Next, the DJI GO application was installed on the Android tablet, and two test flights were conducted. These flights were made on our campus grounds. The flights were documented; keeping track of date and time of flight and flight patterns. Data was recorded using the drone and manually by the researchers to account for all flight events.

[1] https://datfile.net/ (last seen 2016-10-03).
[2] https://datfile.net/csvDownloads.html (last seen 2016-10-03).
[3] https://healthydrones.com (last seen 2016-10-03).

Table 1
Apparatus.

Tool	Description	Utilization
Flight system		
DJI Phantom III Standard	Quadcopter	Serial# CL03021337
DJI remote control	Remote control	Flight control
Mobile device (Nexus 7 (2013) Tablet Android OS V6.0.)	GPS/Heads up display	Navigation
DJI GO App V3.0.0	Real time video connection	Live video feed
Forensic analysis tools		
Forensic workstation	Mac OSX/MS Windows 10	Test bench
Memory Card Reader	Celebrite—UFED S/N 1004922	Hardware write blocker/Memory card reader
DROP Tool V0.1	DAT/Text file tool	Decode DAT files/file correlation
NMAP V7.3	Network scanner	Identify open ports on flight system network
Hex editor version 1.7.7.0	Hex editor	Reverse engineering
IDA Pro V6.8	Dis-assembler	Reverse engineering
FTK Imager Lite V3.1.1	Forensic Imager	Imaging/Hashing
Autopsy 4.1.1	Forensic analysis tool	Cross validation of results
7Zip 16.03	Compression/decompression	Extract select files
Android Backup Extractor	Decompression tool	Extract backup files
JD-GUI V1.0.0	Java source code viewer	View Java source code of .class files
adb (Adroid Debug Bridge)	communication with android system	Communicate mobile device
SQLite Manager V0.8.3.1	Firefox plugin	SQL Lite DB viewer
Tools used for disassembly		
Metric Allen Key set	Size 2 mm	Separate drone body
General Tools 63518 Eighteen-Piece Precision Screwdriver Set	Micro Phillips Flathead, and Torx bits	Circuit board removal, shell separation, and motor removal
Dremel rotary tool	Electric drill	Removing plastic tabs that hold the flight lights in
Knife	Utility knife	Cutting away the glue from the SD card holder

Data acquisition

Imaging external SD card

Once the test flights were completed, the acquisition of data commenced. We began by taking a physical image of the 64 GB external SD card used by the Gimbal camera system to store images and video. With the drone and remote turned off, we extracted the micro SD card from the Gimbal rig and inserted it into a Cellebrite write blocker. The MD5 hash was generated and stored. Then, using the disk dump (`dd`) utility, the entire disk was dumped to an image file and hashes were compared and verified.

Subsequently, the image file was opened in FTKImager 3.1.1 where the content of the image was examined and extracted to another folder (Autopsy 4.1.1 was used to cross-validate findings). The extracted files included images and videos taken throughout the flights as well as some files with metadata about the videos. Results are shared in more detail in Section Testing and findings.

Android backup

Attention was then turned to the Nexus 7 tablet which acted as a control station and dashboard for the drone. It displayed the live camera feed, battery information, GPS, and allowed the user to issue commands such as automated take off and landing. Android backups consist of primarily application related artifacts and typically provide data pertaining to the ownership and usage of the DJI GO application. A logical backup was executed using `adb backup -all` and Android Backup Extractor by Elenkov (2014) was used to extract the resulting `backup.ab` file to a `.tar` file. 7zip was then used to decompress the file to a directory containing the files pertaining to each application installed on the tablet, which included files for the DJI GO application located in apps/dji.pilot. One may also want to create a physical image of the Android device, but it was unnecessary as we were attempting to locate logical data on the device.

Android storage

While the Android backup was useful in acquiring application data, the user data must be obtained through the mobile device's nonvolatile internal storage. To do this, the device was connected to the forensic workstation and all of the files located on the internal storage were selected and copied to the computer. The internal storage contained several directories pertaining to DJI and these were analyzed for relevant artifacts.

Acquiring the drone's storage

The last stage of the data acquisition was to obtain the flight records from the drone's nonvolatile internal storage. This was conducted using three different methods:

1. Mounted the drone's internal storage to the forensic workstation via the DJI GO application and manually copied the files to the forensic workstation. We note that the internal storage was mounted as read only. Our tests showed that this method may not be forensically sound as the drone has to be turned on (Section Acquisition testing).
2. This step is identical to the first except that now we acquire a physical image of the internal storage using `dd`. Note, the same limitations apply.
3. For this method, the actual nonvolatile internal storage medium was pulled from the drone and forensically acquired. This involved disassembling the drone, disconnecting several wires, and cutting away the glue that was intended to permanently hold the internal SD card in place. The socket for the SD card on the drone is shown in Fig. 1. Our tests showed that this was the most forensically sound acquisition method.

Data analysis

There were two primary sources for flight data from the DJI Phantom III. These include TXT files created by the DJI GO mobile application and stored on the mobile device and DAT files created by the drone itself and located on the drone's nonvolatile internal storage. Both files are encrypted and encoded using two different proprietary formats. After decrypting and decoding these files, data regarding the GPS, motors, remote control, flight status, and other information can be extracted. These files essentially serve as the electronic flight recorder for the drone.

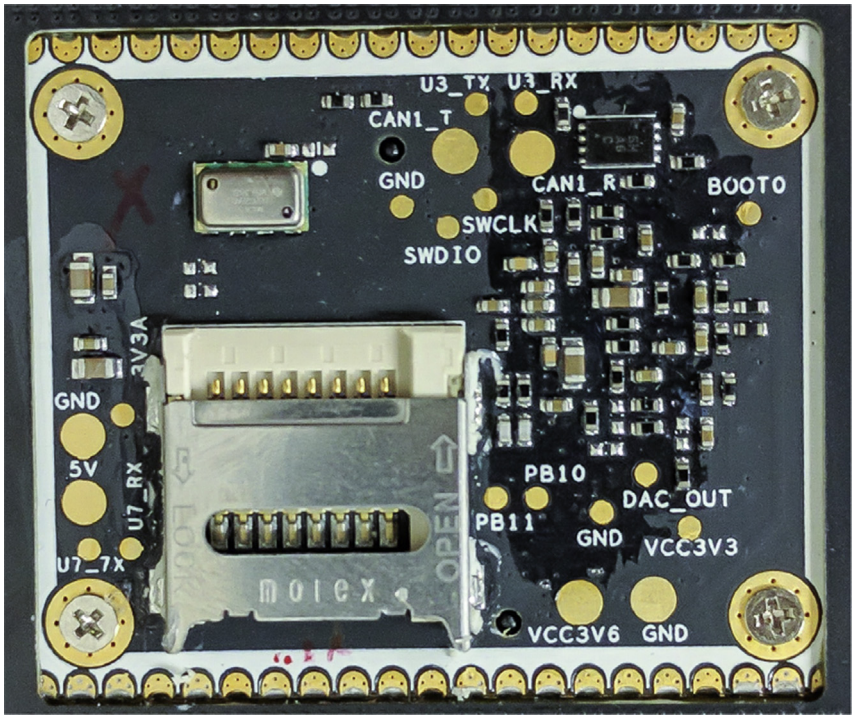

Fig. 1. Extracting the Phantom III internal SD card from the bottom of the drone's main internal PC board.

DJI GO flight record TXT files

While analyzing the files from the Android device's nonvolatile internal storage, several TXT files containing flight data were discovered. These files were found in `InternalStorage/DJI/dji.pilot/FlightRecord/`. The files were named using a standard naming convention of `DJIFlightRecord_YYYY-MM-DD_[HH-MM-SS].txt`. The date and time within the file name corresponds to the date and time that the drone flight began. This file contains data regarding location, flight status, battery levels, and more in the form of packets. These packets must be decrypted and decoded.

DJI GO TXT file structure

Knowing that these text files are created by the DJI GO application we focused on trying to reverse engineer the android application to ascertain the file structure. The official DJI GO application was downloaded[4] and decompiled using JD-GUI.[5] With this program, all the java class files became visible. Although many variables, functions, and class names were obfuscated, key words could be searched for. A search for "FlightRecord" (the name of the directory where the TXT files are stored) pointed us to `k.class` inside the `dji.pilot.fpv.model` package of classes2.jar. `k.class` handled the flight record file writing process. Through manual inspection of the file writing process within the Android application and the TXT file itself using a hex editor, we concluded that the file follows a structure. The high level file structure is shown in Fig. 2. It is important to note that data in this file was in little Endian.

The last 190 bytes of the file contain general information about the aircraft and flight. The first piece of information is an optional geotag in plain text for where the flight took place. For example, many of our files read "New Haven, Connecticut." The next bit of recognizable data comes several bytes later and states the drone's name in plain text as seen here:

```
Yuhe's Phantom3
03Z0600080
CL03021337
05LD102XHR
1153516293
```

This is the name assigned by the user during the set up process along with the model name, where "Yuhe" is the name of the owner of the drone. This is followed by what appears to be four identification numbers. The same four numbers were also found in the `dji_pilot_publics_model_DJIDeviceInfoStatModel` table of `dji.db`. We discovered that these are actually serial numbers and through inspection of the application in JD-GUI, we were able to correlate the serial numbers to hardware devices.

The first number can be traced to the Inertial Measurement Unit (IMU) located inside the drone. Serial number `CL03021337` represents the camera. The third number can be traced back to the primary circuit board inside the remote control that was used to control the drone. This board is actually responsible for transmitting the on-screen data to the mobile device. The last number, `1153516293` belonged to the battery.

TXT packet structure

Flight record data is written to TXT files in the form of packets, the general structure of which is specified in Fig. 3.

The payload of the packet is encrypted and encoded. At the time of writing there was no published method of decrypting this payload, but during the process of reverse engineering the DJI GO Android application, we were able to locate a library `DJI GO_v2.9.1_apkpure.com 2/lib/armeabi-v7a/libFREncrypt.so`, which is a proprietary library used to encrypt and decrypt the flight record data for the TXT files. So far, not much has been published on this library, however, some work has been done trying to reverse engineer it. We have been able to isolate the encryption and decryption functions using IDA-Pro and are currently in the

[4] https://apkpure.com/dji-go/dji.pilot (last seen 2016-10-03).
[5] http://jd.benow.ca/ (last seen 2016-10-03).

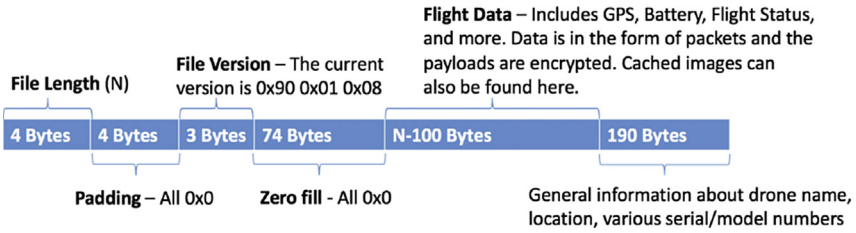

Fig. 2. Breakdown of TXT file structure.

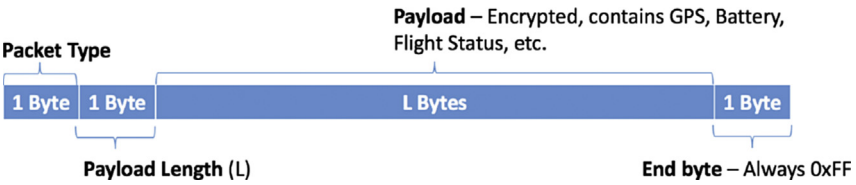

Fig. 3. Breakdown of TXT packet structure.

process of tracing the program stack to understand the input parameters of the function.

DJI drone DAT files

During the work on the drone several proprietary DAT files were discovered on the 4 GB nonvolatile internal storage of the drone. These files followed a common naming convention of `FLY###.DAT`, where the "`###`" is a successive number. This type of file contains a large chunk of flight data related to the drone's location, flight status, and various sensor readings.

DJI DAT file structure

After extracting the files from the drone's internal SD card (FAT32 file system), some preliminary work was conducted in an attempt to read the file. It was quickly determined that the file was encoded and would require decoding. Little official documentation was found related to DJI's DAT files. However, there is a large amount of conversation about these files, as well as several tools created by hobbyists that attempt to decode the files. In particular a tool called *DatCon*[6] yields the most comprehensive output from the files. However, not all fields are decoded. *DatCon* provided a means to parse the information from the binary DAT file and export it to a human readable CSV file. *DatCon* was downloaded as a jar executable and by decompiling the jar file, we were able to gain a more comprehensive understanding of the DAT file structure. The high-level DJI DAT file structure is depicted in Fig. 4.

The overall file layout is fairly simple. The first 128 bytes of the file represent the header. Bytes 16–20 contain the word "BUILD" followed by the date and time of the build. We are uncertain of what this build date actually refers to, but it is suspected that it refers to the date of the last revision made to the file structure. The parsing tool we discuss later actually uses the word "BUILD" as an indicator to make sure that the file being read is in fact a DJI DAT file. Byte 128 is the beginning of the data packets. Data packets are written to the file as packed binary structures which are proprietary to DJI. These packets make up the bulk of the file and then the file ends with a varying number of bytes containing all 0's.

DAT packet structure

The data packets are structures of varying length depending on the type of data being written. Although they are different lengths, they do follow a common structure. Fig. 5 shows the basic structure of the data packet.

It is important to note that the packet length refers to the entire length of the packet, which includes the start byte to the last byte in the payload. The tick number is four bytes representing the internal bus clock tick number and comes after the message byte. The payload can be anywhere from 14 to 245 bytes in length. This is where the major variance is between each packet. The packet type and sub-type correspond to the format of the payload.

Packet payload structure

The packet payload is the part of the packet that contains the actual sensor and telemetry data. The payload can be anywhere from 14 to 245 bytes long and the type of payload is determined by the packet type and sub-type. As this is beyond the paper's scope, a full breakdown of the individual payloads can be downloaded.[7] In Fig. 5 there are nine different packet types they are GPS, motor, home point, remote control, tablet location, battery, attitude, flight status, and advanced battery. The GPS payload is the one we know the most about at this point. It contains the GPS location of the drone as it moves through the flight, altitude, 3-axis acceleration, gyro, velocity, magnetometer, and more. There are four bytes at the end of this payload that appear to contain data, but it has not been determined what that data is. The motor and tablet location packet types are two that have not been seen up to this point. Based on the *DatCon* code, the motor payload contains both the speed and the load of all four motors on the drone. The Phantom III Standard, however, does not report these values, and thus we have not seen these types of packets. The tablet location reports the latitude and longitude of the tablet only during a "Follow Me" mission. This is an auto pilot mode which can be enabled in the DJI GO application where the drone follows you.

The home point of the drone is usually automatically set by the DJI GO application, but the user is also able to manually set this if they wish. Home point coordinates are reported in the home point payload. Remote control status such as throttle, rudder and elevator are recorded by the remote control payloads. There are two types of packets that carry battery information (battery and advanced battery), where one appears to contain information about just the overall battery level and the other reports much more detailed information about the battery capacity, temperature, current, and voltage for the individual cells.

[6] https://datfile.net/index.html (last seen 2016-10-03).

[7] https://github.com/unhcfreg/DROP/blob/master/MesssageStructure.xlsx (last seen 2016-10-03).

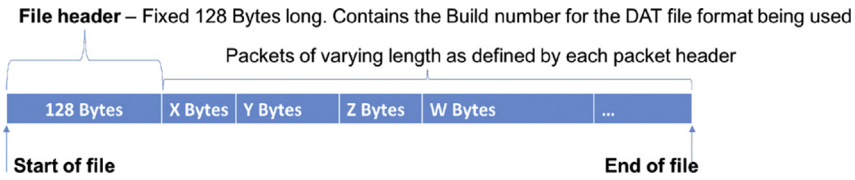

Fig. 4. Basic structure of DJI DAT files.

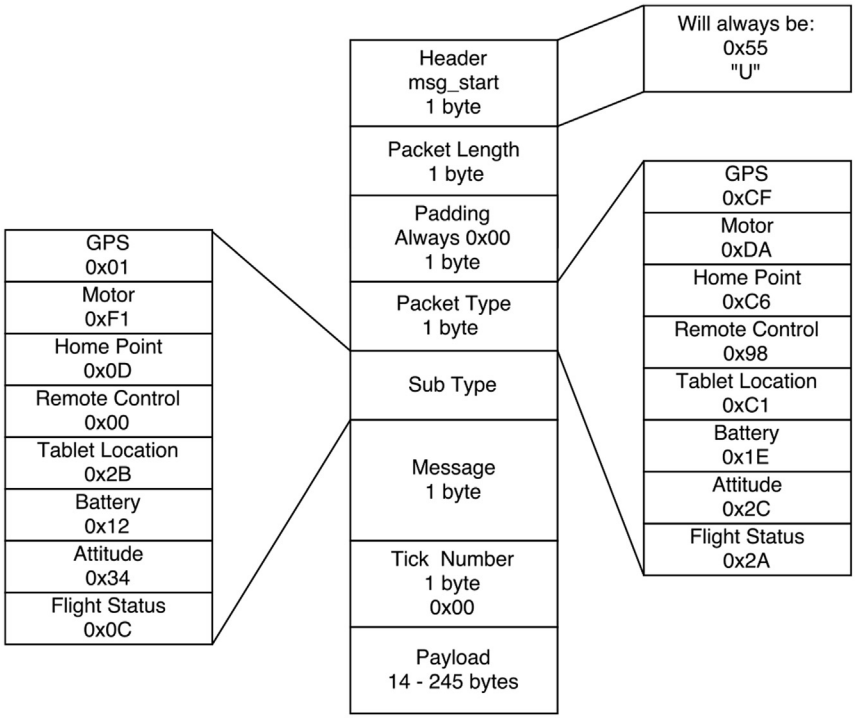

Fig. 5. Breakdown of DAT file structure.

The last two payload types contain data about the Gimbal camera mount and overall flight status. The Gimbal payload contains the positioning of the camera itself. The flight status yields information regarding the state of the flight (autopilot, assisted take off, go home, hover, etc.), and GPS related errors. This packet also contains the flight time in milliseconds since the start of the flight. We demonstrate in the following section that this flight time, along with other metrics, can be used to correlate DAT files to TXT files. This is particularly useful in determining the UNIX time at which events occurred in the DAT file.

Tool creation: DRone Open source Parser (DROP)

DROP, or DRone Open source Parser is a command line forensic tool we constructed in Python 3.4 to parse data from the DJI DAT files. This tool works with Python 3.4 and is largely based on reverse engineering DatCon. It has only been tested on DAT files from the DJI Phantom III Standard, but is expected to also work for the Phantom III Advanced, Professional, and the Inspire 1.

Tool usage

DROP provides two primary functions:

1. Parse data from DJI DAT files
2. Correlate DAT files to TXT files

DROP can be used to process either single DAT files or run a batch of DAT files. This is dictated by the input parameters given at execution. DROP's usage is as follows:

```
python DROP.py input [-h] [-o OUTPUT] [-t T] [-f]
```

The input parameter must be specified as either a directory containing DAT files or a single DAT file. The output parameter `-o` is used to specify the output destination. The `-t` is for specifying a path to a directory containing DJI flight record CSV or a single flight record CSV file. The `-f` flag can be set to indicate that DROP should process the file(s) even if they are not of the standard DJI DAT file type. Lastly, there is a `-h` flag which can be used to print the list of input parameters for quick reference. More information on DROP can be found in the DROP readme file.[8]

Main function – data parsing

Pseudo code for the main function of DROP is depicted in Algorithm LABEL:packetDecode. This method begins by checking the system for file metadata. This is used to obtain the file size. This data and more are logged in a separate text file (not shown) for record keeping purposes. After the metadata is recorded, the input file is opened to determine if it is a DJI DAT file format. It does this

[8] https://github.com/unhcfreg/DROP (last seen 2016-10-03).

by first extracting the 128 byte file header and then checking bytes 16–20 for the word "BUILD". If it finds the word "BUILD" the program continues; otherwise it exits.

If the file is determined to be a DJI DAT file, the program continues by reading the next byte (byte 128) from the file and creates a new `Message` instance. If the file is determined to be of some other type, a counter for non-DAT files is incremented. `message` is an object that represents the current state of the drone by taking in packets and storing the most recent values. The program then enters a loop which terminates at the end of the file. This loop cycles through the remaining bytes in the file, extracting the packet length, header, and payload for each packet and updating `message` with the latest data.

Algorithm 1 DAT packet parsing algorithm

```
1:  procedure DROP(inputFile, outputFile)
2:      meta ← inputFile metadata
3:      in_file ← open inputFile in read binary mode
4:      file_header ← read bytes 0 to 127 of in_file
5:      build ← unpack file_header[16:21] to string
6:      if build != "BUILD" then
7:          exit()
8:      out_file ← open outputFile in write mode
9:
10:     byte ← read next byte from in_file
11:     message ← new Message()
12:     while byte length != 0 do
13:         if byte != 0x55 then
14:             byte ← read next byte from in_file
15:         else
16:             packetLength ← read next byte from in_file
17:             padding ← read next byte from in_file
18:             if padding == 0x00 & packetLength >0 then
19:                 header ← read next 7 bytes from in_file
20:                 payload ← read packetLength - 10 bytes from in_file
21:                 thisPacketTickNo ← unpack header[3:7] to integer
22:                 if message.tickNo == NULL then
23:                     message.tickNo ← thisPacketTickNo
24:                 if thisPacketTickNo != message.tickNo then
25:                     if message.addedData == TRUE then
26:                         out_file ← message.getRow()
27:                         message.addedData ← FALSE
28:                 message.addPacket(packetLength, header, payload)
29:                 byte ← read next byte from in_file
30:             else
31:                 byte ← padding
32:     out_file ← message.getRow()
33:     in_file.close()
34:     out_file.close()
35:     return
```

It was discussed earlier that packets can come in many forms, with the primary difference being the payload. Multiple packets may have the same tick number, meaning they were written to the file at the same time. However, all of the packet types are updated at different intervals, and so it is highly unlikely to see one of every type of packet for a single tick number. The `message` object is updated with the latest data rather than reset so that the output to `outputFile` contains the most up-to-date information for all of the fields for each tick number. This is done using the `addPacket()` method of `message` which handles decoding and parsing the data from the payload. This is further discussed in 6.2.1. In our algorithm, we write a row to `outputFile` each time we see the tick number change, so long as there was data added for that tick number. It is possible (and likely) to see tick numbers skipped all together because packets were read in, but none of them contained data that we know how to parse. If at any point a packet is determined to be fundamentally corrupt (not following the established DJI packet format), a corrupt packet counter is incremented. If a packet is of the correct DJI packet format but contains data we do not know how to parse (unrecognized packet type), then an unknown packet counter is incremented. After the end of the file is reached, the last row is written to `outputFile` and the input and output files are closed. After the processing has completed, various statistics about each file processed are written to `processlog.txt`. This includes statistics such as date and time processed, total number of records, number of corrupt/unrecognized records encountered, and more.

Decrypting and parsing payload data

When data from a new payload is added to `message`, the payload must first be decrypted using the scheme in Algorithm LABEL:payloadDecrypt. The decryption algorithm was observed through reverse engineering of DatCon. This method takes the payload and tick number as input parameters and attempts to decrypt the payload. First, the key must be generated by taking the modulo of the tick number and 256. An empty list is then created which will hold the decrypted version of the payload once finished. Next, each byte in the payload is iterated through. Each byte is XORed with the key that was generated previously, and the result is appended to the decrypted payload list. Once each byte in the payload has been processed, the decrypted version of the payload is returned. It may be observed that the encryption algorithm is a very easy scheme to reverse engineer. At this time, we do not know why DJI chose such a simple algorithm.

Algorithm 2 DAT payload decrypt algorithm

```
1: procedure DECRYPT(payload, tickNo)
2:     xorKey ← tickNo MOD 256
3:     decryptPld ← []
4:     for byte in payload do
5:         append (byte XOR xorKey) to decryptPld
       return decryptPld
```

Once the payload is decrypted, the data can be decoded according to the structures. All of the necessary information is published online in the DROP GitHub repository https://github.com/unhcfreg/DROP. This was done by unpacking the bytes from the payload into their intended data types which may include short, double, float, bytes, and more. Several courtesy calculations are performed at this stage as well. These are calculations that produce values which are not directly available from the payload itself, but are helpful for reference. Examples of this include calculating the watts using the voltage and current, or the total acceleration from the three axis acceleration values.

File correlation

An additional function built in the tool is the ability to correlate the generated TXT files found on the Android mobile device to their corresponding DAT files. This correlation is conducted in two steps accomplished with the algorithm shown in Algorithm 3.

Algorithm 3 DAT to TXT file correlation

```
1:  procedure CORRELATION(dat_data, csv_data)
2:      ft_matches ← 0
3:      gps_matches ← 0
4:      for ft in dat_data do
5:          if ft in csv_data then
6:              ft_matches ← ft_matches + 1
7:              if dat_data[ft][lat] matches csv_data[ft][lat] then
8:                  if dat_data[ft][lon] matches csv_data[ft][lon] then
9:                      gps_matches ← gps_matches + 1
10:     if ft_matches > 0 then return (gps_matches/ft_matches) * 100
        return 0
```

We first identify common flight times that exist in both the TXT file and the DAT file. The flight time is just a measurement of the number of milliseconds that have passed since the drone took flight. Next, for each matching flight time, we compared the GPS coordinates from the same row. If the latitude and longitude values from the DAT file and the TXT file were the same out to 5 decimal places, a counter was incremented. We chose to match out to 5 decimal places because we ran a test using a known correlating DAT and TXT file pair. In this test we calculated the average number of decimal places that the coordinates match on and found that on average they match out to 5.51 decimal places. The standard deviation of these two data sets of GPS coordinates was approximately ± 1.5. Once all of the GPS coordinates have been processed, a final confidence level (as a percentage) is calculated by dividing the number of GPS matches by the total number of flight time matches. This method has been effective for correlating DAT files with TXT files and has been tested for several flights.

This correlation method does pose challenges when the flight pattern and general GPS locations of two separate flights are very similar to each other. One way we can get around this is to add more metrics to correlate on. This involves comparing things like altitude, battery voltage, and number of satellites. We have been able to show that extracted data match nearly one to one for a DAT file and corresponding TXT file. See Appendix B for a graphical representation of data extracted from DAT versus TXT files.

Testing and findings

In this section different tests are performed to explore the forensic soundness of acquisition methods and verify that DROP is forensically sound, and will behave as expected. File hashing was conducted as a basic means to verify files are not modified. Sample corrupt files were sent through the tool as well to examine the tool's robustness. We also tested the effects of a full or missing SD card for the drone's nonvolatile internal storage.

Acquisition testing

The differences in the three acquisition methods in section Data acquisition are subtle, but comparing the results from each provided us with valuable insight about how the drone's nonvolatile internal storage was managed as well as the forensic soundness of each method.

Overall, our results indicated that turning on the drone affect's the integrity of the evidence on the drone's internal storage. Basically, every time the drone is turned on, a new DAT file is created. Therefore, using any acquisition method that required the drone to be on was deemed unsound. Furthermore, we note that other tests were conducted that affect the acquisition of the drone's internal storage. We noticed that if the drone's SD card was at or near full, turning the drone on immediately wiped data (in an irrecoverable manner), thus, affecting the integrity of potentially relevant evidence. The threshold for maximum allowable used space before wiping was not determined during the course of our tests.

File integrity tests

Given that DROP was designed to be used in a potential forensics case it was necessary to verify that the tool did not alter the DAT files in any way. This was accomplished by hashing the files to be parsed before and after they were parsed by DROP.

```
File under test: FLY037.DAT
File hashes:
Library copy: 1804a65d9d97833c6f06adf2e9bd8dbe
After DatCon: 1804a65d9d97833c6f06adf2e9bd8dbe
After DROP: 1804a65d9d97833c6f06adf2e9bd8dbe
```

By observation, it is apparent that the hashes of DAT file FLY037.DAT before and after tool usage match. Both DatCon and DROP were tested for this. Unlike DatCon, DROP automatically calculates and displays the MD5, SHA-1, and SHA-256 hash digests before the file under test is opened and after it is closed. It will also automatically perform the comparison of the hashes and alert the user as to whether or not the hashes matched.

DAT file testing

To test the robustness of DROP and DatCon, we created a set of corrupt DAT files which were all derivatives of one known good DAT file. Thirteen files in total were created, each with unique issues. Tests included removing bytes from the header, changing a packet's start byte value, removing bytes from a payload, and more. Details and results for all of the tests that were performed are shown in Appendix A. This set of tests is comprehensive, but not necessarily complete. More tests could potentially be executed to examine how the software reacts to for example running these tests on every packet in a file, but we deemed this as outside the scope of our testing. Overall, both DatCon and DROP performed equally, with the two exceptions. First, DROP, provided valuable feedback to the user on parsing challenges faced, whereas DatCon did not.

Second, further testing indicated that *DatCon* was missing data when parsing DAT files. *DatCon* will only parse data at a given tick offset time sample frequency. The tick offset time is the number of seconds passed since the beginning of the recording and is calculated using the Central Processing Unit (CPU) tick number. Being that DAT files usually contain an overwhelming amount of data packets, *DatCon* incorporated a feature to allow the user to set the packet sample frequency (packets per second). This effectively adjusts the output resolution of the DAT file. Currently, the fastest sample rate the user can select is 200 Hz. However, even with the frequency set at the maximum possible rate, *DatCon* still fails to output one in five records from the DAT file. Conversely, DROP outputs every packet that is found and does not have a method of setting the sample frequency. This is to say that for every five records outputted by DROP, DatCon will only output four of them.

Drone internal SD card tests

While working with the DAT files we became curious about the limitations of the drone's internal SD card which was formatted as FAT32. Specifically we wanted to explore the effects of filling the SD card to capacity. We began with an inspection of the SD card extracted from the drone. The card had 39 DAT files numbered from 145 to 181. The card had approximately 1182.13 MB of free space. A python script was constructed to analyze the free space left on the card and then create a file of exactly that size in order to fill the card to capacity. With the card at capacity, it was inserted back into the drone and the drone was turned on. After several seconds, it was turned off and the SD card was extracted for analysis. Inspection showed that 12 files were deleted. These were files 145 to 157 which were the oldest DAT files on the card. Moreover, the file pointers were not just deleted, the drive space previously occupied by these files was zeroed out, eliminating all chances of recovery.

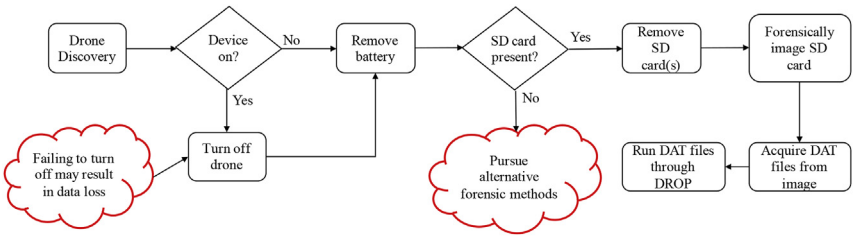

Fig. 6. DJI Phantom III data acquisition process.

Our last test involved flying the drone without the SD card. It was discovered that the drone would in fact fly without an SD card. This may be used as an anti forensics mechanism by adversaries. In an effort to simply and succinctly convey the data acquisition process that should be followed by investigators, we propose the flow diagram in Fig. 6 be used.

DAT and TXT file correlation tests

To externally validate our data parsing method using `DROP`, we chose to use the TXT file parser by healthydrones.com. The idea was to explore if data extracted by our parser on the DAT file (extracted from the actual drone's SD card) matched closely to the data parsed from the TXT file located on the Android device used to control the drone. Our findings indicated that DROP produced results that were almost a one-to-one match as shown in Appendix A. As shown in the graphs, the data from both the TXT files as well as the DAT file follow similar trends. The standard deviation for these three correlations were found to be near zero and can be seen in Table 2. The cause of the differences between DAT and TXT files has yet to be determined.

Forensically relevant findings

Forensic analysis of the DJI Phantom III yielded valuable forensic data such as GPS locations, WiFi connections, user information, dates and times etc., each extracted from a variety of data sources. Table 3 summarizes forensically relevant artifacts.

Table 2
DAT and TXT file correlation standard deviations.

Metric	Standard deviation
Latitude & Longitude	0.00000499
Number of satellites	0.555
Battery voltage	0.0849

Limitations

As discussed at the beginning of this paper in Section Research scope, our research focused on just the DJI Phantom III. We realize that this work does not encompass a complete understanding of forensics for all consumer drones, but it does provide a good starting point. DJI currently holds the largest market share in the US for drone sales, thus validating our focus. Furthermore, based on the amount of research and time involved with performing a thorough forensic analysis, it is currently unrealistic to try and cover all or even a majority of consumer drones. All we can do at this point is to continue research in the area of drone forensics as well as push manufacturers to agree on a standard for recording flight data.

In regards to our tool, `DROP`, the biggest limitation is that both DAT files and TXT files are proprietary. We have not captured the full spectrum of potential data that may be extracted from DAT files, and as such, is a limitation of this work.

Conclusions & future work

While much work has been conducted with regards to security of drones, little work has been published with regards to the forensic analysis of drones. As these devices continue to grow in capabilities it will become necessary to have a forensic method for acquisition and analysis grounded by science and robust testing. The work accomplished in Section Methodology begins to develop this process. It shows that it is possible to identify locations and times of the drone, along with additional forensically relevant data of value to a potential case. The methods developed in this research were scrutinized for forensic soundness. Furthermore, our results

Table 3
Summary of forensically relevant findings.

Finding	Description	Utilization
General findings		
External MicroSD card (64 GB)	Gimbal memory	Photo & video
Internal MicroSD card (4 GB)	Drone main board removable memory	Flight stats
Mobile device	Flight system feedback	Autopilot, pictures, GPS
External MicroSD		
Pictures/	/DJI_####.jpg	EXIF data (date/time, pitch, roll and yaw of Gimbal and aircraft)
Videos/	/DJI_####.m4v, /idx##	Metadata (file headers contained location of drone and GPS data)
Internal MicroSD		
DAT files/	/FLY###.DAT	Flight information (GPS, compass, battery, etc)
Nexus Tablet		
`\apps\dji.pilot\db\dji.db`	Pertinent flight and user information	No-fly zones, user email addresses, last known home point
`\apps\dji.pilot\sp\dji.pilot.xml`	device ID	device serial numbers, last flight location
`\InternalStorage\DJI\dji.pilot\FlightRecord`	DJIFlightRecord_YYYY-MM-DD_[HH-MM-SS].txt	Flight information (GPS, compass, battery, etc), user name, device serial numbers
`\com.android.providers.settings\f\flattened-data`	Credentials/SSID	WiFi credentials, user home address via access point identification
`\Internalstorage\DJI\dji.pilot\CACHE_IMAGE`	Images/EXIF data	Cached images from MicroSD on gimbal assembly

showed that we can link a specific drone based on its serial number to a mobile device that is controlling it.

While our work focused on the DJI Phantom III, more work needs to be conducted across the spectrum of drones available for consumers today, including the recently released Phantom IV. Future work should focus on attempting to demystify the DAT and TXT file structures. Furthermore, our team did attempt to reverse engineer the firmware on the drone, but more work needs to be pursued on that front. This could shed some light on data this work was not capable of parsing. Lastly, it would be of relevance to develop a tool to map location over time given a starting point and accelerometer data from the DAT files stored on the drone. More work should also be conducted on other areas that may hold data of evidentiary value located on the drone, especially since our work showed that potential adversaries are capable of flying the drone even after removing its nonvolatile internal storage.

Acknowledgments

Cinthya Grajeda Mendez – for her help on the preliminary forensic analysis of the Phantom III drone.

A. Correlation data DAT vs. TXT

- See Fig. 7 for GPS coordinates correlation.
- See Fig. 8 for satellite correlation.
- See Fig. 9 for battery correlation.

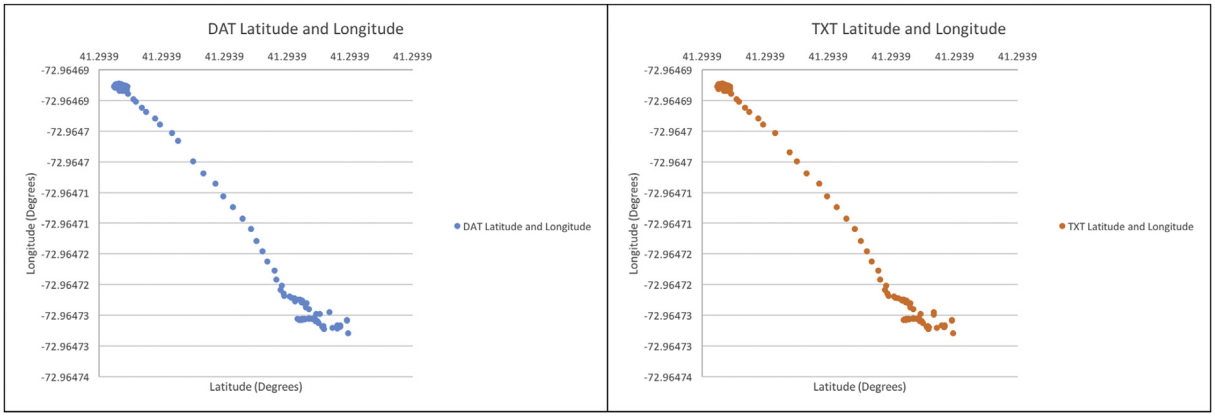

Fig. 7. Latitude Longitude DAT vs. TXT.

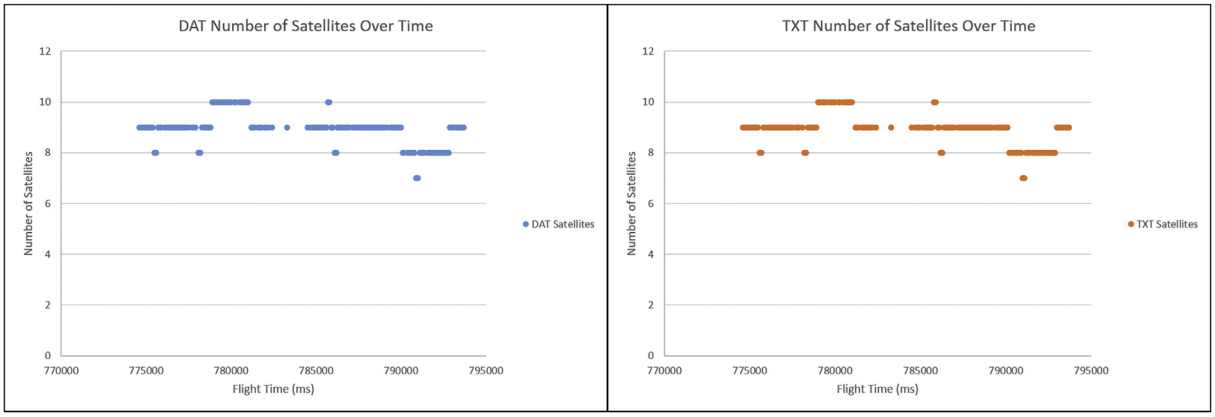

Fig. 8. Satellites DAT vs. TXT.

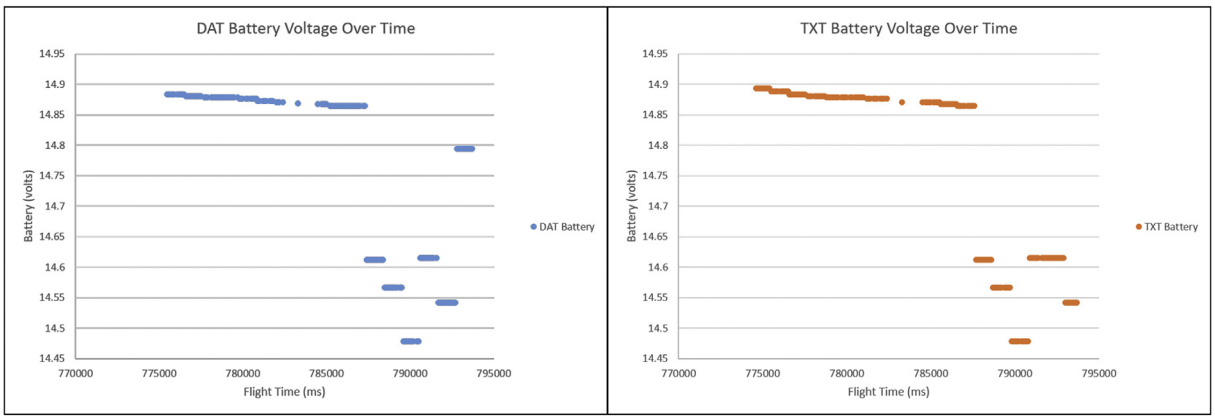

Fig. 9. Battery DAT vs. TXT.

B. Corrupt data test results

- See Table 4 for corrupt data testing information.

Table 4
Corrupt data test results.

File description	DROP results	*DatCon* results
Original file	Runs as expected.	Runs as expected.
Removed 790 bytes starting at byte 128 offset from the beginning of the file.	Runs as expected.	Runs as expected.
Modified "BUILD" in file header – removed it completely	still processed if the force flag has been set, but it starts looking for packets at the beginning of the file. Output matches original.	Completely refused to process the file.
Short File. Removed from byte 7271872 to the end.	DROP processed the data up to record 970326 (The last existing record).	Missed one record at the end (due to sample frequency).
Removed bytes 65–93 from header.	Processed the file but missed the data in the first packet (tick number 970323).	Processed the file but missed the data in the first packet (tick number 970323).
Changed the start byte of message 970326 to 0x66.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and output 0's for this tick number.
Changed message type from CF 01 to CF 32 of message 970326.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and output 0's for this tick number.
Changed packet length from 84 to 67.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and output 0's for this tick number.
Changed packet length from 84 to 92.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.
Changed padding in packet header from 0x00 to 0xAB	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Seems to have worked fine, but hash of output does not match that of the original output.
Removed packet header completely	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and output 0's for this tick number.
Removed bytes 36–73 of the payload (1709–1746 overall)	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.
Added 10 random bytes to end of payload: `59 4F 55 47 4F 54 48 41 4B 44`	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and output 0's for this tick number.
Added 10 random bytes to start of payload: `59 4F 55 47 4F 54 48 41 4B 44`	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.	Processed the file but missed data for record 970326 GPS data and did not output any data for this tick number.
Created a completely random payload	Processed the file but output bad data for record 970326.	Processed the file but output bad data for record 970326.

References

Amato, A., 2015. Drone Sales Numbers. http://dronelife.com/2015/04/16/drone-sales-numbers-nobody-knows-so-we-venture-a-guess/ (Accessed 3 October 2016).

Brandom, R., 2016. Drone Incedents 2016. http://www.theverge.com/2016/3/25/11306850/faa-drone-airport-incidents (Accessed 3 October 2016).

Carver, R., 2014. State drone laws: a legitimate answer to state concerns or a violation of federal sovereignty. Ga. St. UL Rev. 31, 377.

Clarke, R., Moses, L.B., 2014. The regulation of civilian drones' impacts on public safety. Comput. Law Secur. Rev. 30, 263–285.

Elenkov, N., 2014. Android Backup Extractor. https://github.com/nelenkov/android-backup-extractor (Accessed 3 October 2016).

Franklin, M., 2016. dji-log-parser. https://github.com/mikeemoo/dji-log-parser (Accessed 3 October 2016).

Horsman, G., 2016. Unmanned aerial vehicles: a preliminary analysis of forensic challenges. Digit. Investig. 16, 1–11.

Kaminski, M.E., 2013. Drone Federalism: Civilian Drones and the Things They Carry.

Kamkar, S., 2013. SkyJack. https://github.com/samyk/skyjack [Accessed 3 October 2016].

Kiltz, S., Lang, A., Dittmann, J., 2007. Taxonomy for computer security incidents. Cyber Warf. Cyber Terror. 412–417.

Kovar, D., 2015. Drone Forensics. https://files.sans.org/summit/Digital_Forensics_and_Incident_Response_Summit_2015/PDFs/ForensicAnalysisofsUASakaDronesDavidKovar.pdf (Accessed 3 October 2016).

Luo, A., 2016. Drones Hijacking. https://media.defcon.org/DEF\%20CON\%2024/DEF\%20CON\%2024\%20presentations/DEFCON-24-Aaron-Luo-Drones-Hijacking-Multi-Dimensional-Attack-Vectors-And-Countermeasures-UPDATED.pdf (Accessed 3 October 2016).

Ravich, T.M., 2015. Courts in the drone age. N. Ky. L. Rev. 42, 161.

Rondinone, M., 2015. Drone Gun. http://www.courant.com/breaking-news/hc-gun-fire-drone-investigation-20150721-story.html (Accessed 3 October 2016).

Samland, F., Fruth, J., Hildebrandt, M., Hoppe, T., Dittmann, J., 2012. Ar. drone: security threat analysis and exemplary attack to track persons. In: IS&T/SPIE Electronic Imaging. International Society for Optics and Photonics, 83010G–83010G.

Schmidt, S., 2016. ISIS: Exploding Drones. https://www.nytimes.com/2016/10/12/world/middleeast/iraq-drones-isis.html?_r=0 (Accessed 24 January 2017).

Thiobane, F., 2015. Cybersecurity and Drones (Ph.D. thesis). UTICA COLLEGE.

Wild, G., Murray, J., Baxter, G., 2016. Exploring civil drone accidents and incidents to help prevent potential air disasters. Aerospace 3, 22.

Zenko, M., 2012. 10 things you didnt know about drones. Foreign Policy 192, 62–63.