Hindawi
Security and Communication Networks
Volume 2017, Article ID 1308045, 9 pages
https://doi.org/10.1155/2017/1308045

Research Article

Automatic Reverse Engineering of Private Flight Control Protocols of UAVs

Ran Ji, Jian Wang, Chaojing Tang, and Ruilin Li

College of Electronic Science and Engineering, National University of Defense Technology, Changsha 410073, China

Correspondence should be addressed to Jian Wang; jwang@nudt.edu.cn

Received 15 March 2017; Revised 14 May 2017; Accepted 29 May 2017; Published 4 July 2017

Academic Editor: Zheng Yan

The increasing use of civil unmanned aerial vehicles (UAVs) has the potential to threaten public safety and privacy. Therefore, airspace administrators urgently need an effective method to regulate UAVs. Understanding the meaning and format of UAV flight control commands by automatic protocol reverse-engineering techniques is highly beneficial to UAV regulation. To improve our understanding of the meaning and format of UAV flight control commands, this paper proposes a method to automatically analyze the private flight control protocols of UAVs. First, we classify flight control commands collected from a binary network trace into clusters; then, we analyze the meaning of flight control commands by the accumulated error of each cluster; next, we extract the binary format of commands and infer field semantics in these commands; and finally, we infer the location of the check field in command and the generator polynomial matrix. The proposed approach is validated via experiments on a widely used consumer UAV.

1. Introduction

1.1. Background and Protocol Reverse Engineering. The use of civil unmanned aerial vehicles (UAVs) has increased considerably in recent years, largely due to the innovations, reasonable price, and diverse capabilities of such UAVs [1, 2]. At present, civil UAVs are being widely deployed in such applications as aerial photographs, plant protection operations in agriculture, and border patrolling. However, a UAV can also be used for inappropriate applications. UAV abuse cases, such as flight safety regulations and no-fly area violations, pose a growing threat to public safety and privacy [3]; such cases have been reported by mass media and have gained increasing attention from the public. Therefore, airspace administrators urgently need an effective method to regulate UAVs.

Reverse analysis of flight control commands in the communication channel between the UAV and controller helps us understand the meaning and format of these commands and is highly beneficial to the regulation of UAVs. However, a number of flight control protocols are undocumented or private, that is, having no publicly available specifications. Thus, it is difficult to reverse-analyze UAV control commands because of their private design.

Automatic protocol reverse-engineering [4] techniques enable private protocols to be understood. The goal of automatic protocol reverse engineering is to extract the grammar, semantics, and state machine of private protocol messages [5–7]. Thus, we can use automatic protocol reverse-engineering techniques to reverse analyze UAV flight control commands.

1.2. Related Work. Many researchers have published valuable studies on automatic protocol reverse engineering. There are two main approaches to reverse-analyze private protocols automatically: based on dynamic program analysis and based on network traces [8]. Polyglot [9] and Dispatcher [10] use dynamic binary program analysis to extract the message format of private protocols. Dispatcher can also extract the field semantics of protocol commands. However, the flight control programs of target UAVs are not always available for airspace administrators. Methods based on dynamic program analysis (such as Polyglot and Dispatcher) suffer from certain limitations. For example, they require the flight control programs. Different from Polyglot and Dispatcher, Protocol Informatics [11] uses algorithms found in the bioinformatics field to extract the format of protocol messages collected from network traces. Shevertalov and Mancoridis proposed a tool based on network traces called PEXT [12] to analyze

finite state machines using private protocols. In addition, PEXT uses the longest common subsequence (LCSS) [13] to classify private protocol messages into clusters. Discoverer [14] leverages recursive clustering and type-based sequence alignment to infer private protocol message formats from its network trace. PRE-Bin [15] extracts bit-oriented formats of protocol message formats from its network trace based on hierarchical clustering, the multiple sequence alignment algorithm, and the Bayes decision model. Li and Yu [16] mined keywords of private protocols tunneled over WebSocket using a hidden semi-Markov model [17] based on network traces. However, methods proposed in previous work are often unavailable to the scientific community [18]. Netzob [18] is an open-source protocol reverse-engineering tool for analyzing protocol message formats and protocol state machines. This protocol features high accuracy when analyzing complex communication protocols.

Approaches based on network traces mentioned in [11, 12, 14–16, 18] focus on network communication protocols such as ICMP, FTP, DNS, or certain undocumented communication protocols running on botnets instead of UAV flight control protocols. As a result, the approaches described above cannot be used for reverse engineering of private flight control protocols for UAVs directly, and they cannot extract the meaning of flight control commands. However, their fundamental concepts are creative and valuable. Thus, we can use these concepts when reverse-analyzing flight control protocols.

To the best of our knowledge, few publications have described protocol reverse engineering of the private flight control protocols of UAVs. Rodday et al. [19] proposed a method to analyze the meaning of flight control commands by decompiling the Android app that is offered to control UAVs. However, the method will be invalid if the app is unavailable.

1.3. Main Contribution and the Outline. To address the issues we noted above, this paper designs an automatic reverse analysis method based on network traces for the private flight control protocols of UAVs. First, the flight control commands collected from the binary network trace are classified into clusters by length and LCSS [13]. Then, the meanings of the flight control commands are inferred based on the fact that UAV movement is a real-time reaction to received flight control commands. Then, we extract the binary format of flight control commands and analyze the semantics of a variable field in these commands. Finally, we infer the location of the check field and the generator polynomial matrix, which is used to calculate the value of the check fields.

In summary, this paper makes the following contributions: the method is applicable to analyze the flight control commands of UAV. The method can extract the meaning of the commands, parse the format of flight control commands, infer the semantics of fields in these commands, and determine how to generate the check field (CRC or other codes).

The proposed method is based on binary network traces and does not require prior knowledge regarding the target flight control protocol or the flight control programs of the target UAV. However, the target UAVs must be visible when inferring the meaning of flight control commands.

This paper is organized as follows: a brief introduction to UAVs flight control commands is provided in Section 2. Section 3 proposes the automatic reverse analysis method for the private flight control protocols of UAVs. In Section 4, the proposed approach is validated via experiments on a widely used consumer UAV (names and models are not disclosed because of a nondisclosure agreement). Section 5 presents a concise summary of this paper.

2. Terminology

UAVs are typically operated using a remote controller or relevant app running on a smart phone to send commands to the UAVs. UAVs perform various actions, such as taking-off, landing, rotating, flying forward, and flying backward, according to the commands. Our approach uses a set of conversations collected from binary wireless network traces as input. Conversations consist of commands exchanged between UAVs and a remote controller. The commands can be classified into clusters by properties such as length or keywords. Certain commands are used to keep the conversation alive or indicate acknowledgment for certain commands but do not move UAVs, whereas other commands operate UAVs to move vertically or horizontally.

In general, as shown in Figure 1, a command is composed of a sequence of fields that contain a number of bits and has diverse values. Fields of commands with a constant value in a cluster are frequent in every command and are referred to as keywords. Keywords reflect the format of a group of commands and determine the meaning or function of commands to a certain extent. Fields with variable values are referred to as dynamic fields. A dynamic field typically has semantics, such as sequence number or the value of the check field (i.e., check bits). For different flight control protocols, the locations of keywords and dynamic field may be different from those shown in Figure 1.

To be more specific, an open flight control protocol of UAVs, MAVLink [20], is introduced to help understand flight control command. Each MAVLink flight control command is 17 bytes long and has the structure shown in Figure 2. Fields 1 and 2 are keywords with constant value; other fields are dynamic. Semantics of each field are listed in Table 1. The command function depends on the message ID.

3. Methodology

There are four steps in our method: (1) classify the flight control commands collected from the binary network trace into clusters, (2) analyze the meaning of the flight control commands, (3) extract the format of the commands and infer the field semantics in the commands, and (4) infer the location of the check field and the generator polynomial matrix. The architecture of the system is illustrated in Figure 3.

3.1. Classify Commands into Clusters. Because the formats of commands are fixed for a given flight control protocol, types of commands share the same structure, where keywords are fixed and the dynamic field values vary in a certain range

Table 1: Semantics of each field in MAVLink flight control command.

Field	Type	Length in bytes	Semantic	Value
1	Keyword	1	Message header	0 × FE
2	Keyword	1	Payload length	0 × 09
3	Dynamic field	1	Sequence number	Rolls around from 255 to 0
4	Dynamic field	1	System ID	Dynamic
5	Dynamic field	1	Component ID	Dynamic
6	Dynamic field	1	Message ID	Dynamic (e.g., 0 = heartbeat)
7	Dynamic field	9	Payload	Dynamic (depending on message ID)
8	Dynamic field	2	Checksum	Dynamic

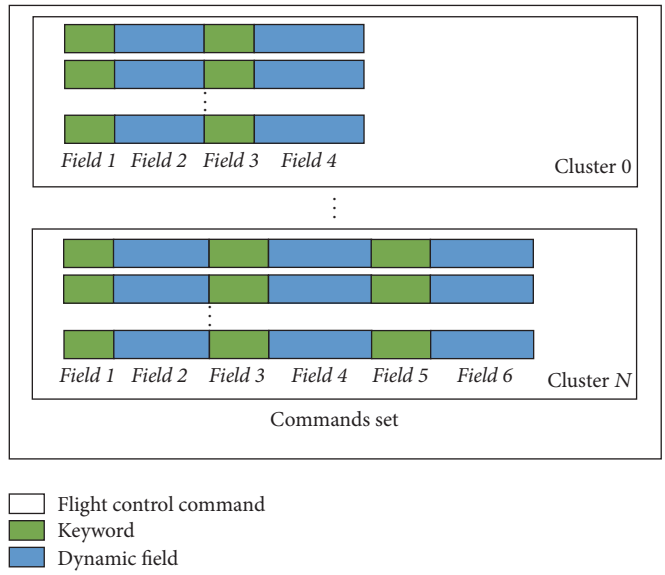

Figure 1: Examples of commands composed of a sequence of fields in conversations in a UAV flight control link.

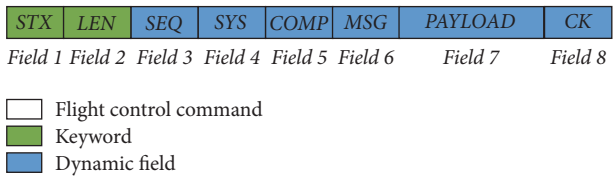

Figure 2: MAVLink [20] flight control command structure.

but the length of the dynamic field is constant. We classify flight control commands into clusters using two steps. First, we group commands into clusters depending on the length of the commands. Then, in cases in which different types of commands have the same length, we group commands of the same cluster into subclusters according to a certain distance among commands. We use the LCSS [13] described in [12] to calculate the distance between two commands with the same length, a and b, and we use hierarchical clustering [21] to classify the cluster into subclusters based on the LCSS distance.

$$D(a,b) = 1 - \frac{\text{length}(\text{LCSS}(a,b))}{\max(\text{length}(a), \text{length}(b))}. \quad (1)$$

3.2. Analysis of the Meaning of Flight Control Commands. Because the speed of wireless signal propagation in airspace is close to the speed of light and the operation speed of the MCU (microcontroller unit) in UAVs is extremely fast, the movement of UAVs is approximately a real-time reaction to corresponding flight control commands. Therefore, the intervals between adjacent moments when a given UAV begins to perform a given action are consistent with the intervals between the adjacent moments when the corresponding command is received by the UAV. As a result, the deviation between these two types of intervals should be

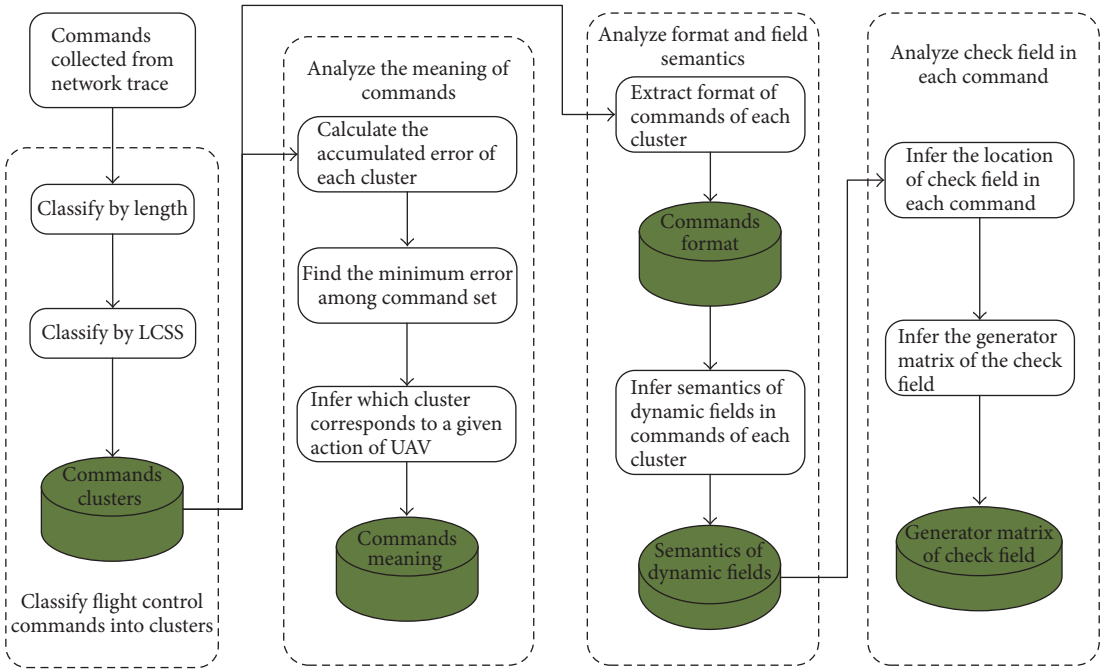

Figure 3: System architecture.

extremely small, and the intervals between adjacent moments when other commands are received by the UAV do not have consistency with intervals of that action.

We denote the set of flight control commands collected from a binary network trace as

$$S = \{C_i \mid i = 0, 1, 2, \ldots, N\}, \tag{2}$$

where C_i is the ith cluster in the command set.

Cluster C_i is denoted as a sequence of commands; that is,

$$C_i = \{m_{ij} \mid j = 0, 1, 2, \ldots, M\}, \tag{3}$$

where m_{ij} is the jth command in cluster C_i.

To acquire the meaning of the flight control commands, we tag each message m_{ij} with a timestamp t_{ij} to record the time when the message is received. We also record the time T_i^Γ when the target UAV begins to perform a given action Γ, such as taking-off or landing, for the ith time and the target UAV performs the action $\Gamma K + 1$ times. If the number of commands in a cluster is less than $K+1$, then these commands in the cluster cannot be corresponding commands for the action Γ. Thus, we do not consider them when we infer commands the cluster contains that control the target UAV to perform action Γ.

We consider two types of time interval sequences:

(1) *UAV Action Time Interval Sequences* Δ^Γ. Each time interval in this sequence indicates the time differences between adjacent moments when the UAV begins to perform the same action Γ; that is,

$$\Delta_k^\Gamma = \{T_{k+1}^\Gamma - T_k^\Gamma \mid k = 0, \ldots, K-1\}. \tag{4}$$

(2) *Receiving Message Time Interval Sequences* TC_{ij}. Each time interval in this sequence indicates the time differences between adjacent moments when message m_{ij} in cluster C_i is received; that is,

$$TC_{ij} = \{t_{ij+1} - t_{ij} \mid j = 0, \ldots, M-1\}. \tag{5}$$

We define the accumulated error E_{Ci}^Γ of cluster C_i, which measures the deviation between Δ^Γ and TC_{ij}:

$$E_{Ci}^\Gamma = \sum_{j=0}^{M-K} \left(\sum_{k=0}^{K-1} \left| TC_{ij+k} - \Delta_k^\Gamma \right| \right). \tag{6}$$

Assuming that the minimum accumulated error E_{Ci}^Γ for action Γ among all clusters C_i $(i = 0, 1, 2, \ldots, N)$ in the set of commands collected for network trace S is E_{CL}^Γ, we can infer that the commands in cluster C_L are the commands controlling the target UAV to perform action Γ. If several values of E_{Ci}^Γ represent the lowest accumulated errors, the commands belonging to the cluster whose amount is near $K+1$ are corresponding commands that are more relatively likely. The meaning of flight control commands reverse analysis algorithm is summarized in Algorithm 1 for clarity.

3.3. Extract Format and Infer Field Semantics. We utilize the *Protocol Informatics* project [11] to extract the binary format of commands. This method applies the sequence alignment algorithm to extract formats. The details of this method have been published in [11] and are thus not presented here.

Whereas keywords reflect the format or structure of a group of commands, the field semantics of dynamic fields

Input: Action Γ beginnning time T_k^Γ $(k = 0, 1, \ldots, K)$
Timestamp t_{ij} of command m_{ij} $(i = 0, 1, 2, \ldots, N,\ j = 0, 1, 2, \ldots, M)$
Output: Commands in which the cluster control target UAV to do action Γ,
Initiation: min = 0, all $E_{Ci}^\Gamma = 0$ $(i = 0, 1, 2, \ldots, N)$, temp = 0
(1) **for** all k **except** $k = K$ **do**
(2) $\quad \Delta_k^\Gamma = T_{k+1}^\Gamma - T_k^\Gamma$
(3) **end for**
(4) **for** all i **do**
(5) $\quad$ **for** all j **except** $j = M$ **do**
(6) $\quad\quad TC_{ij} = t_{ij} + 1 - t_{ij}$
(7) $\quad$ **end for**
(8) **end for**
(9) **for** all i **do**
(10) $\quad$ **for** $j \leftarrow 0$ to $M - K$ **do**
(11) $\quad\quad$ temp = 0
(12) $\quad\quad$ **for** $k \leftarrow 0$ to $K - 1$ **do**
(13) $\quad\quad\quad$ **add** $|TC_{ij+k} - \Delta_k^\Gamma|$ **to** temp
(14) $\quad\quad$ **end for**
(15) $\quad\quad$ **add** temp **to** E_{Ci}^Γ
(16) $\quad$ **end for**
(17) **end for**
(18) **print** Commands in cluster min control the target UAV to do action Γ

Algorithm 1: The meaning of commands reverse analysis algorithm.

describe the type of command. Typical dynamic field semantics include the sequence number, check field, and length of message. Different field semantics have different attributes. For example, the sequence number increases or decreases progressively over time. The check field is always located at the end of a command, and its value does not have a clear trend, unlike the sequence number. Field semantics are fundamental to understanding the function of a message, and we will demonstrate how to infer the location of the check field and reverse-analyze the generator polynomial matrix of the check field in Section 3.4.

3.4. Inferring the Location of the Check Field and Generator Polynomial Matrix. The check field is always located at the end of a command. The method employed here considers that if the last dynamic field in a given command does not exhibit a clear trend, it is likely the check field. The generator matrix is essential when analyzing how to generate the value of the check field. There are many possible ways to calculate the check field value, including a cyclic redundancy check (CRC) and a parity check. However, for a private protocol, we do not have prior knowledge about the checking technique, the generator matrix, or the checking range, that is, the input when calculating the value of the check field. We propose a method to reverse-analyze how the target UAV generates check values and what the generator matrix is in affine space without the aforementioned prior knowledge. We also validate the obtained generator matrix by comparing the constructed check values with those collected from the network trace.

The format of a typical command Com is shown in Figure 4. The command consists of 4 fields: keyword K_1, dynamic field D, keyword K_2, and check field C. These fields are L_1, L_2, L_3, and L_4 bits long, respectively. We consider the command Com as a block matrix. Thus, fields K_1, D, K_2, and C are blocks in Com, with sizes of $1 \times L_1$, $1 \times L_2$, $1 \times L_3$, and $1 \times L_4$, respectively. We assume that the generator matrix is G. Consequently, G can be partitioned into three blocks: G_1, G_2, and G_3. The sizes of G_1, G_2, and G_3 are $L_4 \times L_1$, $L_4 \times L_2$, and $L_4 \times L_3$, respectively. G, K_1, D, K_2, and C satisfy (9). The only operation when calculating the value of the check field is logical XOR.

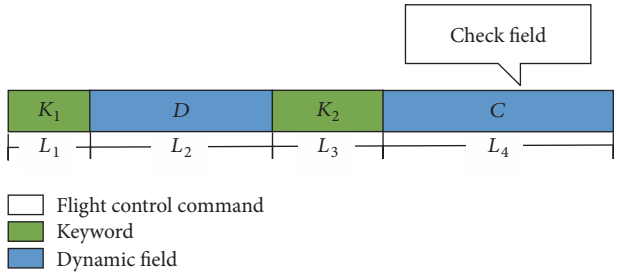

Figure 4: Typical command Com format in a UAV flight control link.

$$\text{Com} = [K_1 \ \ D \ \ K_2 \ \ C], \tag{7}$$

$$G = [G_1 \ \ G_2 \ \ G_3], \tag{8}$$

$$[G_1 \ \ G_2 \ \ G_3] \begin{bmatrix} K_1 \\ D \\ K_2 \end{bmatrix} = C. \tag{9}$$

Input: A cluster C_{Com} consists of the given command Com
Format of the command or the cluster
new dynamic field $D(k)$ of a new command belongs to cluster C_{Com}
Output: Vector B and generator matrix is $G2$ of the cluster
Check field value of the new command corresponding to $D(k)$
Initiation: $N \leftarrow$ length of check field, SP $\leftarrow$ a set of all possible N bits binary sequence
(1) **for** each bit in check field **do**
(2) flag = 0
(3) **for** *item in SP* **do**
(4) **for** (Com i, Com j) in C_{Com} **do**
(5) **if not** equation (12) **then**
(6) flag = 1
(7) **break**
(8) **end if**
(9) **end for**
(10) **if** flag == 1 **then**
(11) **break**
(12) **end if**
(13) join *item* in $G2$
(14) **end for**
(15) **end for**
(16) Com $s \leftarrow$ an arbitrary command in cluster C_{Com}
(17) $B \leftarrow C^{(s)} \oplus G2D^{(s)}$
(18) Check value $\leftarrow B \oplus G2D(k)$

Algorithm 2: Generator matrix analysis and check of the field reconstruction algorithm.

The kth bits in the check fields of commands Com i and Com j are assumed to be $C_k^{(i)}$ and $C_k^{(j)}$, respectively, because they satisfy (9); $C_k^{(i)}$ and $C_k^{(j)}$ also satisfy

$$G_{1,k}K_1 \oplus G_{2,k}D^{(i)} \oplus G_{3,k}K_2 = C_k^{(i)}, \tag{10}$$

$$G_{1,k}K_1 \oplus G_{2,k}D^{(j)} \oplus G_{3,k}K_2 = C_k^{(j)}, \tag{11}$$

where $G_{1,k}$, $G_{2,k}$, and $G_{3,k}$ are the kth rows in G_1, G_2, and G_3, respectively.

$G_{1,k}K_1 \oplus G_{3,k}K_2$ is denoted as B_k. The XOR of (10) and (11) is

$$G_{2,k}\left(D^{(i)} \oplus D^{(j)}\right) = C_k{}^{(i)} \oplus C_k{}^{(j)}. \tag{12}$$

Because $D^{(i)}$, $D^{(j)}$, $C^{(i)}$, and $C^{(j)}$ are known, we can solve each row in generator matrix G_2 using (12). Then, we can solve each bit in B based on (10) or (11). For any command Com l with a new dynamic field $D^{(l)}$, we can calculate its check value by

$$C^{(l)} = B \oplus G_2 D^{(l)}. \tag{13}$$

Equation (13) helps us reconstruct the check field for a new dynamic field. We can validate the obtained generator matrix by comparing the reconstructed check field with the one collected from a network trace. The check field reverse analysis and reconstruction algorithm is summarized in Algorithm 2 for clarity.

Other command formats with different numbers of keyword fields and data fields can also apply the reverse analysis and reconstruction method.

4. Evaluation

In this paper, we use a consumer UAV that has a considerable market share in the civil UAV market as the reverse analysis target to demonstrate the validity of our method. The name and model of the UAV are not disclosed due to a nondisclosure agreement. The UAV has a WPA2-protected [22] WiFi link between itself and the operator's smartphone. The operator can operate the UAV takeoff and landing by an official app running on the smartphone. As the TCP payload, the flight control commands are transmitted by a TCP link between the UAV and smartphone. We crack the WiFi link by dictionary attack [23] and intercept the binary network trace of the TCP link and extract the TCP payload from TCP segments as the dataset for our experiment. We collect 132,966 TCP segments by applying a Wireshark and 802.11n wireless USB adapter. Because the direction of commands is from the smartphone to UAV, we preprocess the TCP segments to extract payloads sent by the smartphone as the command set under consideration. The total number of commands is 10,672. We implement the proposed method by Python on Linux OS with a 3.4 GHz CPU.

It requires 1.79 s to classify the commands into clusters. As shown in Table 2, we classify the commands that we collect from the network trace into 11 clusters. Cluster Number 10 might belong to TCP ACK segments because they do not have a data segment. Clusters Number 0 and Number 4 only have one message; thus, we do not extract their keywords. Clusters Number 6 and Number 7 have different keywords but the same length.

Table 2: Clusters of UAV flight control commands.

Cluster number	Length in byte	Keyword(s) in hexadecimal form	Amount
Number 0	35		1
Number 1	22	551604FC020B, 00533000000000000000000	56
Number 2	20	5514046D020482074	4
Number 3	19	551304030201DE0640026A01	2
Number 4	18		1
Number 5	17	551104920203, 4003F8713D65CB	4
Number 6	14	550E04660228, 400804	372
Number 7	14	550E04660207, 80000E	142
Number 8	13	550D04330208DD0640083164F2	2
Number 9	1	FF	140
Number 10	0		9948

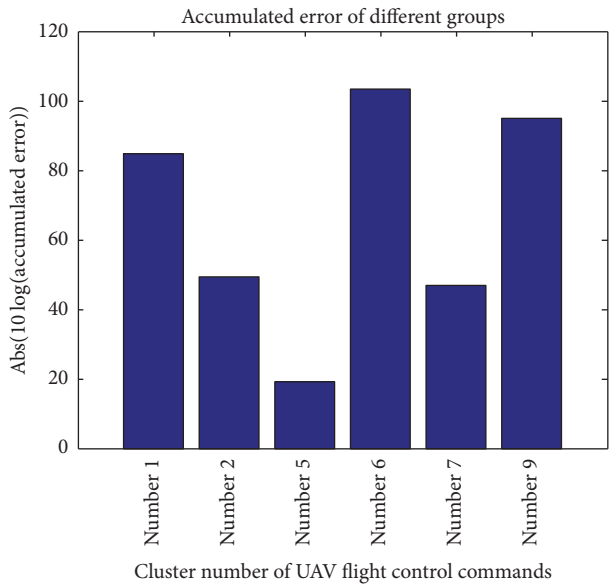

Figure 5: Accumulated error of each cluster.

Figure 6: Format of the takeoff command.

In the experiment, we extract the command that controls UAV takeoff. The UAV takes off at four times—at 9:10:02 am, 9:10:52 am, 9:11:42 am, and 9:12:32 am. The time intervals between the four takeoff actions are 50 s. It requires approximately 0.22 s to extract the target command. Based on Algorithm 1, we infer that the commands in Cluster Number 5 control the UAV takeoff. As shown in Figure 5, we calculate the accumulated error E_{Ci}^{Γ} (Γ = takeoff) of each cluster C_i ($i = 0, 1, \ldots, 9$). Clusters Number 0, Number 3, Number 4, and Number 8 cannot consist of the takeoff commands because their commands amounts are less than 4, that is, $K+1$ in (4). Thus, we do not show their accumulated error. The accumulated error E_{C5}^{Γ} of Cluster Number 5 is the lowest of all of the clusters in the dataset.

Then, we extract the binary format of the takeoff commands using *Protocol Informatics* [18], which requires 0.03 s. In addition to the keywords shown in Table 1, there are two dynamic fields in the command, as shown in Figure 6.

We collect another 100 takeoff commands to infer the field semantics precisely. The variety pattern of the dynamic field in these commands indicates that the first dynamic field is a sequence number because its values increase progressively over time (see Figure 7) and the second dynamic field is the check field because it is located at the end of the commands and does not have a clear trend (see Figure 8).

Algorithm 2 is used to reverse-analyze the way in which the UAV generates the check field and generator matrix in affine space. We use 15 of these 104 takeoff commands to solve the generator matrix and the other 89 commands to verify that the generator matrix is correct. It requires approximately 45.79 s to acquire the generator matrix. A comparison of the reconstructed check field with the remaining check fields collected from the network trace yields an accuracy of 100%. One of the generator matrices (generator matrix $G2$ of the takeoff command) is shown in (14). The check field is generated by CRC16, because the generator matrix with a cyclic shift property and the length of the value in the check field is 16 bits.

$$\begin{bmatrix}
1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 1 \\
1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\
1 & 0 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\
0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \\
1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 & 1 & 1
\end{bmatrix} \quad (14)$$

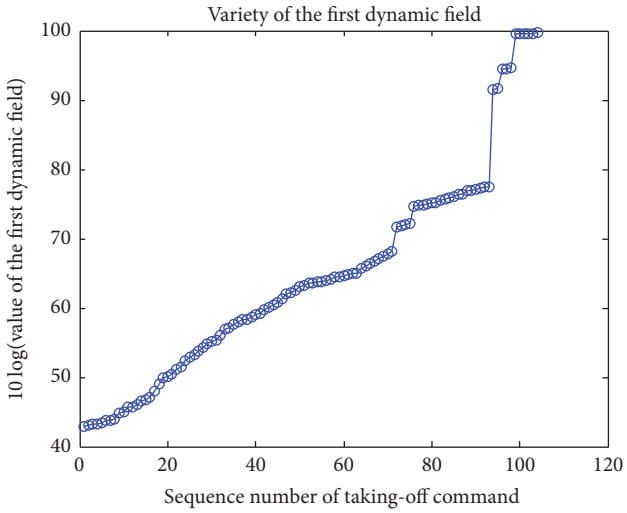

Figure 7: Values in the first dynamic field.

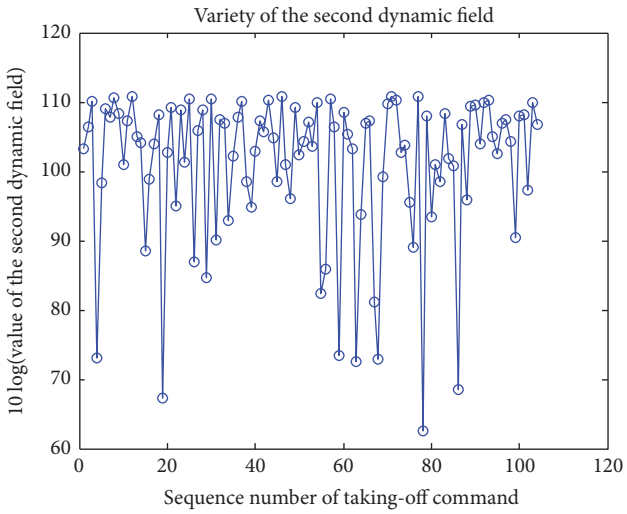

Figure 8: Values in the second dynamic field.

5. Conclusion

Understanding the meaning and format of UAV flight control commands by automatic protocol reverse-engineering techniques is highly beneficial to UAV regulation. This paper proposed a method to analyze the private flight control protocols of UAVs. First, we classify flight control commands collected from a binary network trace into clusters; then, we analyze the meaning of the flight control commands by the accumulated error of each cluster; next, we extract the format of the commands and infer the field semantics in these commands; and finally, we infer the location of the check field and the generator polynomial matrix that generates the check field. In experiments, we reconstructed the flight control command successfully. Future work will aim to leverage computer vision technology to improve the method of mining the meaning of flight control commands.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] K. A. Demir, H. Cicibas, and N. Arica, "Unmanned aerial vehicle domain: areas of research," *Defence Science Journal*, vol. 65, no. 4, pp. 319–329, 2015.

[2] J.-S. Pleban, R. Band, and R. Creutzburg, "Hacking and securing the AR.Drone 2.0 quadcopter: investigations for improving the security of a toy," in *Proceedings of the IS & T/SPIE Electronic Imaging International Society for Optics and Photonics*, San Francisco, Calif, USA, February 2014.

[3] R. L. Finn and D. Wright, "Unmanned aircraft systems: surveillance, ethics and privacy in civil applications," *Computer Law and Security Review*, vol. 28, no. 2, pp. 184–194, 2012.

[4] J. Narayan, S. K. Shukla, and T. C. Clancy, "A survey of automatic protocol reverse engineering tools," *ACM Computing Surveys*, vol. 48, no. 3, article 40, 2015.

[5] W. Ying, L.-z. Gu, Z.-x. Li, and Y.-x. Yang, "Protocol reverse engineering through dynamic and static binary analysis," *Journal of China Universities of Posts & Telecommunications*, vol. 20, 2, pp. 75–79, 2013.

[6] M.-M. Xiao, S.-L. Zhang, and Y.-P. Luo, "Automatic network protocol message format analysis," *Journal of Intelligent and Fuzzy Systems*, vol. 31, no. 4, pp. 2271–2279, 2016.

[7] J. Antunes, N. Neves, and P. Verissimo, "Reverse engineering of protocols from network traces," in *Proceedings of the 18th Working Conference on Reverse Engineering, WCRE 2011*, pp. 169–178, irl, October 2011.

[8] X. Li and L. Chen, "A survey on methods of automatic protocol reverse engineering," in *Proceedings of the 2011 7th International Conference on Computational Intelligence and Security, CIS 2011*, pp. 685–689, chn, December 2011.

[9] J. Caballero, H. Yin, Z. Liang, and D. Song, "Polyglot: automatic extraction of protocol message format using dynamic binary analysis," in *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pp. 317–329, ACM, November 2007.

[10] J. Caballero, P. Poosankam, C. Kreibich, and D. Song, "Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering," in *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS '09)*, pp. 621–634, ACM, Chicago, Ill, USA, November 2009.

[11] M. Beddoe, "Network protocol analysis using bioinformatics algorithms, 2009".

[12] M. Shevertalov and S. Mancoridis, "A reverse engineering tool for extracting protocols of networked applications," in *Proceedings of the 14th Working Conference on Reverse Engineering, WCRE 2007*, pp. 229–238, can, October 2007.

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, McGrawHill, 2nd edition, 2001.

[14] C. Weidong, J. Kannan, and H. J. Wang, "Discoverer: Automatic protocol reverse engineering from network traces, 2007".

[15] S. Tao, H. Yu, and Q. Li, "Bit-oriented format extraction approach for automatic binary protocol reverse engineering," *IET Communications*, vol. 10, no. 6, pp. 709–716, 2016.

[16] B.-C. Li and S.-Z. Yu, "Keyword Mining for Private Protocols Tunneled over WebSocket," *IEEE Communications Letters*, vol. 20, no. 7, pp. 1337–1340, 2016.

[17] S.-Z. Yu, "Hidden semi-Markov models," *Artificial Intelligence*, vol. 174, no. 2, pp. 215–243, 2010.

[18] G. Bossert, F. Guihéry, and G. Hiet, "Towards automated protocol reverse engineering using semantic information," in *Proceedings of the 9th ACM Symposium on Information, Computer and Communications Security (ASIA CCS '14)*, pp. 51–62, 2014.

[19] N. M. Rodday, R. O. De Schmidt, and A. Pras, "Exploring security vulnerabilities of unmanned aerial vehicles," in *Proceedings of the 2016 IEEE/IFIP Network Operations and Management Symposium, NOMS 2016*, pp. 993-994, tur, April 2016.

[20] qgroundcontrol.org. MAVLink Micro Air Vehicle Communication Protocol, 2017, http://qgroundcontrol.org/mavlink/startfrom.

[21] O. Maimon and L. Rokach, *Data Mining and Knowledge Discovery Handbook*, Springer, New York, NY, USA, 2005.

[22] Wi-Fi Alliance. WIFI Direct, 2017, http://www.wi-fi.org/discover-wi-fi/wi-fi-direct.

[23] wikipedia, org. Dictionary attack, 2017, https://en.wikipedia.org/wiki/Dictionary_attack.