Diversity: A Poor Man's Solution to Drone Takeover

Ali Davanian[1], Fabio Massacci[2] and Luca Allodi[3]
[1]*Redsocks security, Laan van Nieuw, The Hague, Netherlands*
[2]*Department of Information Engineering and Computer Science, University of Trento, Povo, Trento, Italy*
[3]*Faculty of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, Netherlands*

Keywords: Micro Air Vehicle Communication Protocol (MAVLink) Security, Drone Takeover, Unmanned Aerial Vehicle (UAV) Security, Moving Target Defense (MTD), Software Diversity.

Abstract: Drones are the new targets for hackers. On one hand, their widespread use and security weakness have made them very attractive for the attackers. On the other hand, there are a few security solutions for drones. Out of these few, some are just proposals, and fewer are in the early stage of development. We first assess the requirements of a security solution for drones and then analyse the effect of traditional cryptographic solutions on the drone's traffic volume and energy consumption. With recourse to Moving Target Defence, we propose a novel instruction diversity solution for drone security that is portable, and has zero overhead.

1 INTRODUCTION

In recent years, Unmanned Aerial Vehicles (UAV) also known as drones have gained a lot of attention especially for commercial purposes (UAV Expert News, 2016); yet less has been done toward the security of drones. In a recent Proof of Concept (POC), a police drone has been successfully hijacked (Rodday, 2016). In future attempts, adversaries can go even further and covertly take over the drones. An adversary can mislead the operator and mount a stealthy takeover attack by using the communication protocol predefined commands. A stealthy takeover especially applies to the radio-based drones where the drone is out of the sight.

MAVLink is the most famous UAV protocol (Figure 1). Security weakness of MAVLink protocol (Meier et al., 2011) is well known among UAV community (Google groups, 2013). The protocol neither incorporates authentication nor encryption. Several solutions have been proposed for more recent versions of MAVLink protocol (Meier et al., 2013) (Tridgell and Meier, 2015). These solutions however are not fully implemented. In addition, MAVLink structural limits and performance constraints propelled others to abandon MAVLink and develop custom protocols like GIDL (Pike, 2013).

We propose a 'poor man's' security solution for MAVLink with recourse to Moving Target Defense (Evans et al., 2011) and software diversity (Larsen et al., 2014). This solution offers reasonable security against general purpose and targeted attacks whilst imposing zero performance overhead. In the rest of this paper, in section 2, we shed light on the drone takeover attacks and the existing security solutions for MAVLink. In section 3, we present the threat model for UAV systems and future stealthy-automatic version of the takeover attacks. In section 4, we discuss the performance constraints that security solutions must consider for UAV systems. In section 5, we present our poor man's solution to the drone takeover and enhance it based on our security analysis. In section 6, we conclude the paper.

2 ATTACKS AND DEFENCES

WIFI and Radio drones have proved to be vulnerable since a long time ago. Deligne was the first scholar who could reverse engineer a toy drone and leverage its lack of authentication (Deligne, 2012). Deligne circumvented a simple Mac Filtering by spoofing the MAC address of the Ground Control Station's (GCS). One year later, Kamkar presented a drone capable of hacking other drones (Kamkar, 2013). This drone, named Skyjack, exploits MAVLink lack of authentication vulnerability. Skyjack hacks into other drones by using aireplay-ng and sending deauth packets to disconnect the legitimate owners. Since the drone does not authenticate the users, the attacker can

Davanian, A., Massacci, F. and Allodi, L.
Diversity: A Poor Man's Solution to Drone Takeover.
In *Proceedings of the 7th International Joint Conference on Pervasive and Embedded Computing and Communication Systems (PECCS 2017)*, pages 25-34
ISBN: 978-989-758-266-0

take over as soon as the WIFI connection is established. In the same year, the security of MAVLink over SIK Radio modem has been analyzed (Marty, 2013); similarly, because of lack of MAVLink authentication, an attacker can take over if she can establish the Radio connection to the drone. Marty assumes that the initial modem configuration, to communicate with the drone, is already known. Because of the broadcast nature of Radio communication, the attacker just requires the NET ID of the drone and the GCS for the attack. With a SIK Radio modem, this can be easily obtained by changing the modem's behavior to communicate with every node using the sniffed Net ID (Shellntel, 2017). Even without sniffing, the Net ID can be brute forced because it is only 16 bits long.

Rodday's work (Rodday, 2016) is the most recent work in the drone takeover area. Rodday acquires the target MAC Address using a XBEE broadcast message. The XBEE protocol family follows IEEE 802.15.4 standard (Layer, 2013). The drone on which Rodday mounts his attack works at 868 MHz frequency. According to IEEE 802.15.4 standard, the topology can be either MESH or Peer-To-Peer. With the former, the attacker needs to find the Personal Area Network (PAN) ID while with the latter, the node's destination address is necessary for the communication. Rodday's contribution is in finding the drone's destination address using a simple broadcast command. Finding MAC Address by a bruteforce is also possible due to its prognostic structure; vendor ID is part of the MAC address and the attacker can guess this. After the drone responds with its destination address, the attacker can send any commands she wishes.

Drones cannot accept commands from multiple sources since "During a mission at most one 'Navigation' command and one 'Do' or 'Condition' command can be running at one time" (ARDUPILOT, 2017b). In order to overcome this limitation, Deligne, Kamkar and Rodday rely on deauthenticating the WIFI user. Since such vulnerability does not exist in XBEE, the aforementioned Proof of Concepts assume, the owner is not active in case of Radio hijacking. That said, in practice, the attacker need disconnect the legitimate user, and consider the user reaction when the latter finds out her drone is hijacked. Otherwise, the user intervenes physically when the drone is nearby e.g. in case of WIFI drones (150 meters coverage) or the attack fails because the drone receives commands from two sources e.g. in case of Radio drones. As a result of such constraint, adversaries will mount "stealthy takeover attacks".

To address the threats against drones, the community has proposed new secure drone-operator communication protocols. The first notable secure UAV project based on MAVLink protocol is SMACCMPilot (Pike et al., 2013). SMACCMPilot invokes GIDL as the application level protocol. GIDL adapts MAVLink and uses AES GCM for authentication and encryption. This solution's drawback, however, is portability and compatibility since it is based on Haskell backend. 3DR, Yuneec, Parrot and many other vendors use MAVLink itself and already have their own infrastructure.

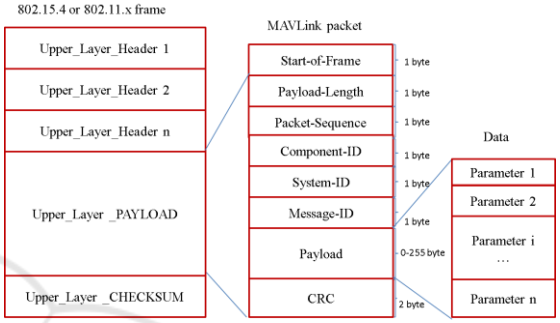

Figure 1: MAVLink packet encapsulation.

Meier et al., proposed a secure version of MAVLink protocol (Meier et al., 2013) with compatibility constraint in mind. sMAVLink has the same encryption algorithm as GIDL though it is not based on Haskell. While GIDL encrypts the payload, header and CRC, sMAVLink focuses only on the payload. This allows the architect to leave the MAVLink packet structure untouched. That said, to support the solution, the payload (or message) structure should evolve according to Figure 2.To the best of our knowledge, sMAVLink has never been implemented.

Since January 2016, MAVLink version 2.0 supports Packet Signing as a security feature (Tridgell and Meier, 2015). A fast authentication mechanism with backward compatibility is the motivation of packet signing solution. MAVLink packet signing is the result of prolonged discussion between MAVLink and Pixhawk (a well-known autopilot implementation) developers (Google groups, 2013). Key agreement in packet signing is done offline via a USB or Serial cable (similar to GIDL), and the key is 32 bytes long. The key is used to compute the hash of the legacy packet and a few new fields. These new fields are link-id (8 bits), timestamp (48 bits) and the signature itself (48 bits). Link-id is the identifier of a communication channel between GCS and the drone. Timestamp is used to avoid replay attacks. The signature is the first 48 bits of a SHA-256 hash of the

secret key, header, payload, CRC, link ID and timestamp. These 13 bytes (including signature, link ID and timestamp) should be added to the end of a MAVlink packet. For backward compatibility, an environment parameter in firmware is used to determine whether to accept unsigned packets. Whenever this packet signing parameter is zero, packets without signature are accepted. This parameter is not communicated during the flight.

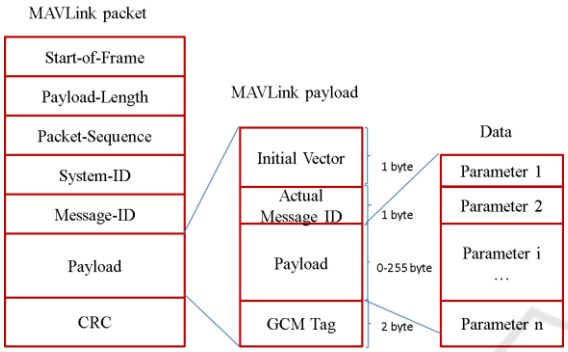

Figure 2: sMAVLink payload structure.

Backward compatibility and portability is a problem with all the aforementioned solutions. GIDL is the worst solution in this regard because it is built on a complete new infrastructure. sMAVLink has never been implemented probably because it completely changes the structure of MAVLink packets and imposes many portability issues. Although packet signing is backward compatible and best portable options among all available solutions, it requires the vendors to update their source codes to MAVLink version 2. Given its recent release time, it takes time before vendors can adapt their branches and update to the last version of MAVLink to support packet signing. In summary, a practical solution to address the current security problems immediately must have *backward compatibility*, *portability* and *implementation simplicity* requirements.

3 THREAT MODEL

We identify a taxonomy of attackers in the spirit of (Anderson and Kuhn, 1996):

- Script kiddies
- Motivated attackers
- Funded organizations

Script kiddies are attackers who attack just for fun or thieves who target a drone for its payload value. Script kiddies usually do not create a tool or an exploit; rather, they use off-the-shelf tools. Aircrack-ng (Rodday, 2016) (Kamkar, 2013), killerbee (Spears et al., 2011) and MAVProxy (Marty, 2013) are some of the tools used for the proof of concepts discussed in section 2 Since all these tools are python-based, they may be integrated in a global downloadable tool to automate the whole attack.

Motivated attackers have an adequate knowledge of the system. Attackers from this group target a specific drone or a set of drones, *perform motivated* reconnaissance before attacking and spend time analysing the system. They not only use the same tools script kiddies use but also customize them for their target based on their analysis. They can monitor the target for a longer period.

A *funded organization* consists of professional adversaries working as a team. These professionals have not only the skills to analyze the system meticulously but also adequate funding that allows them to recruit insiders from the vendor companies. Such attackers, for example, may lure an employee to install a drone specific backdoor like Maldrone (Sasi, 2015) on the drone. Attackers of this type are outside our threat model.

Currently available attacks, as described in section 2, are oversimplified and not applicable in practice. A more powerful attack that all actors may perform in future is a *stealthy takeover attack* that we define in the following way:

1. Victim is identified
2. Drone is attacked over the air
3. Original GCS is disconnected
4. The operator observes no abnormality
5. The drone flies to the attacker's nest

Step 3 and 4 are essentially what make a drone takeover successful in a real case scenario. Otherwise, the user intervention makes the takeover impossible. In a successful stealthy takeover, the attacker sends falsified monitoring data to hide the attack from the user. These misleading monitoring data prevent the user from any subsequent counteraction (incident response) that may hamper the attack. Figure 3 illustrates a schema of a stealthy takeover attack. Table 1 reports the MAVLink commands that implement each step described in Figure 3. Importantly, to utilize these commands for any MAVLink based drone, the attacker can *fully automate* its exploit messages using MAVLink header files prepared by Lorenz Meier et al (GitHub, 2017a).

```
1.  Initialize(Connection_Parameters)
2.  FootPrint(Target)
   a) Grab_Model_Info(Drone_Address)
   b) Find_FeedBack_Interval(Drone_Model)
   c) Gather_TargSet_Trajectory(Master_
    adrs)
3. Take_Over_Drone(Drone_Adrs,GCS_Adrs)
   a) Cut_Off_Telemetry(GCS_Address)
   b) Mislead_GCS(GCS_Address, Interval,
    Trajectory)
   c) Send_Control_Command(Drone_Adrs)
```

Figure 3: Stealthy take over attack.

Table 1: MAVLink predefined commands needed for stealthy attack.

Phase. Step	Generic Command	MAVLink Command
2.1	WHO_ARE_YOU	AUTOPILOT_VERSION
2.2	FEEDBACK_INTERVAL	MAV_CMD_GET_MESSAGE_INTERVAL
2.3	NEXT_WAYPOINT	MISSION_CURRENT
2.3		MISSION_REQUEST
2.3		MISSION_REQUEST_LIST
3.1	DISCONNECT_GCS	CHANGE_OPERATOR_CONTROL
3.1		MAV_CMD_COMPONENT_ARM_DISARM
3.1		MAV_CMD_SET_MESSAGE_INTERVAL
3.1		MAV_CMD_DO_SET_MODE
3.2	DRONE_FEEDBACK	HEARTBEAT
3.2		GPS_RAW_INT
3.2		GPS_RAW
3.2		GLOBAL_POSITION_INT
3.2		SYS_STATUS
3.3	GO_TO_LOCATION	MAV_CMD_NAV_WAYPOINT
3.4	LAND	MAV_CMD_NAV_LAND

4 PERFORMANCE CONSTRAINT

Any security solution for drones should cope with their performance constraints. Performance constraint limits latency, persistent space and power consumption overhead. A security solution imposes latency in two ways. Firstly, it increases processing time. For instance, MAVLink packet signing has a computational overhead of 26 microseconds per packet (Tridgell and Meier, 2015). Secondly, a security solution adds extra bytes to every packet for the purpose of verification. This will increase the network traffic and the communication time. Moreover, a security solution will increase the firmware size. For example, MAVLink packet signing code is 812 bytes. The code size is not negligible given PIXHawk flash size that is 1 MB (Tridgell and Meier, 2015). Last but not the least, power consumption increases because of the increase in CPU and network usage.

To understand the effect of an encryption oriented solution, we assess network traffic and protocol power consumption overhead of a security solution. In order to define Protocol Energy Consumption overhead (E_o), we define Δ_E (measured in μJ) as the difference of protocol energy consumption after(E_a) and before (E_b) applying a security solution. Moreover, we define traffic overhead (T_o) in terms of extra traffic bytes per command (Δ_T) divided by the original command traffic size. Δ_T (in byte) is the difference of the traffic size per command after applying a security solution (T_a) and traffic size per command before security control in place(T_b):

$$E_o = \frac{\Delta_E}{E_b} \qquad \Delta_E = E_a - E_b \tag{1}$$

$$T_o = \frac{\Delta_T}{T_b} \qquad \Delta_T = T_a - T_b \tag{2}$$

Total energy consumption E is modeled as a linear composition of a data transmission energy consumption function, $E_{Tr}(.)$ and security control energy consumption function, $E_S(.)$:

$$E = E_S(.) + E_{Tr}(.) \tag{3}$$

$E_S(.)$ models the required energy to calculate the cypher from the message and verify the message using signature. In other words, E_S depends on the message size (T_n), number of messages (m) and the signature size (S). T_n is the total size of the protocol's headers and the payload for the layer n that the security solution is defined. We extrapolate E_S as a linear function of the required energy to encrypt/decrypt a byte (E_{SB}):

$$E_S = E_S(T_n, S) \cong m * (T_n + S) * E_{SB} \tag{4}$$

$E_{Tr}(.)$ models the required energy to transmit messages. E_{Tr} depends on the number of messages (m), total number of bytes to be transmitted for each message (T) and transmission energy consumption per byte (E_{tB}):

$$E_{Tr} = m * T * E_{tB} \tag{5}$$

In order to calculate T, we need to calculate the actual number of required bytes to transmit one message of layer n. We assume a communication protocol like MAVLink works on layer n. We assume the maximum payload size to be sent over the network on

the ith layer of a protocol stack is p_i byte. T_i , which is total size of message in ith layer, consists of the T_{i+1} payload and ith-layer headers, h_i. Depending on the fragmentation, T_{i+1} will be sent using α_i chunks of size p_i . T, total traffic, is T_1 i.e. it is a message with all the additional headers on the physical media:

$$\begin{cases} T_n = P_n + H_n \\ T_i = T_{i+1} + \alpha_i H_i \qquad 1 \le i < n \end{cases}, \quad (6)$$

T_{i+1} must fit into α_i chunks of size p_i. The number of fragmentations α_i must thus satisfy the following conditions:

$$\begin{cases} \alpha_n = 1 \\ \alpha_i = argmin((\alpha_i - 1)\, p_i < T_{i+1} \le \alpha_i p_i), \\ \qquad 1 \le i < n \end{cases} \quad (7)$$

Considering that layer 1 is purely physical and just propagates layer 2 frames ($H_1 = 0$), we can calculate T:

$$T_1 = T_2 + 0 = \sum_{i=2}^{n-1} H_i\, \alpha_i + \alpha_2 p_2 \quad (8)$$

Example: We consider the combination of MAVLink and XBEE and application of AES and SHA1 encryption algorithms as security controls. We base our calculation on transmission energy consumption reported in (Fourty et al., 2012) and cryptographic algorithms energy consumptions reported in (Potlapally et. al, 2003).

In MAVLink case, n is equal to 3 since there is no transport protocol layer between MAVLink and Xbee. We call α_2 after applying encryption $\alpha_{2,a}$ and α_2 before applying encryption $\alpha_{2,b}$. Hence:

$$T_a = H_2\alpha_{2,a} + p_2\alpha_{2,a} = \alpha_{2,a}(H_2 + p_2)$$
$$T_b = H_2\alpha_{2,b} + p_2\alpha_{2,b} = \alpha_{2,b}(H_2 + p_2)$$
$$\Delta_E = m * ((\alpha_{2,a} - \alpha_{2,b}) * (H_2 + p_2) * E_{tB} + (T_3 + S) * E_{SB})\,.$$

Packet size (T_3) for MAVLink version 1.0 is 263 bytes (Figure 1). As packet signing redesigns MAVLink, T_3will increase to 276 bytes. We can assume other encryption based security solutions have at least the same increase for T_3. Moreover in XBEE, header size can vary depending on the mode (DIGI, 2017). For the sake of this study, we assume XBEE does not operate in encryption mode. Table 2 summarizes the value of the variables for MAVLink over XBEE.S for MAVLink Packet Signing is 13 bytes and for sMAVLink is 15 bytes (refer to section 2). Solving (7) using Table 2 variables, we deduce $\alpha_{2,b}$ and $\alpha_{2,a}$value. According to $\alpha_{2,a}$ values, sMAVLink in broadcast mode has T_o of 33% while T_o for other cases is 0 since no additional frames are needed.

Table 2: This table reports the number of required frames before applying encryption $\alpha_{2,b}$and the number of frames after applying encryption $\alpha_{2,a}$to send a message based on the datalink layer protocol and security status.

Mode	Unsecure version		Signature ≤ 13		13<Signature ≤ 60	
Broadcast mode	Variable	Value	Variable	Value	Variable	Value
	T_3	263	T_3	276	T_3	277 – 336
	T_2	100	T_2	100	T_2	100
	H_2	8	H_2	8	H_2	8
	$\alpha_{2,b}$	3	$\alpha_{2,a}$	3	$\alpha_{2,a}$	4
Peer to peer mode	T_3	263	T_3	276	T_3	277 – 336
	T_2	100	T_2	100	T_2	100
	H_2	16	H_2	16	H_2	16
	$\alpha_{2,b}$	4	$\alpha_{2,a}$	4	$\alpha_{2,a}$	4

Limiting T_3 by $\alpha_{2,a}$ using (7), we conclude that if $S \le 13$, there is no overhead in traffic. On the other hand, if $13 < S \le 60$, T_o can increase to 33%. Since AES and AES GCM are in the same family, we approximate S to be in the same range as sMAVLink ($13 < S \le 60$). We also approximate S of SHA1 to be ≤ 13 since MAVLink packet signing uses SHA family (SHA256).

Table 3: It reports the protocol energy consumption overhead per command ($\Delta_E$and E_o) based on S.

	S	E_{tB}	E_{SB}	Δ_E	E_o
AES	15	7.09 μJ	1.62 μJ	1.159 mJ	55%
SHA	13	7.09 μJ	0.76 μJ	0.210 mJ	10%

Even if the traffic overhead is zero, there is at least 10% energy consumption overhead in addition to the processing time overhead. In (Fourty et al., 2012), transmission energy consumption of an 802.15.4 network has been analyzed. For a 19 bytes length frame, the energy consumption is 134.85 μJ. Moreover, from (Potlapally et. al, 2003), we can derive an approximation of the energy consumption of AES (for 128 bit length key and based on CBC block cypher) and SHA1. AES and SHA1 energy consumption per byte are reported respectively 1.62 μJ and 0.76 μJ. We calculate Δ_E per byte for AES and SHA1 based on S value we approximated. Table 3 presents energy consumption values per byte.

5 DIVERSITY SOLUTION

Software diversity, a branch of moving target defense, as a protection mechanism has gained a lot of attentions in recent years (Larsen, et al. 2014). The idea in diversification is to produce final machine (or intermediate) codes that fulfil the same purpose though they are different in form. Instruction Set Diversity is a diversification method that is used to thwart code injection (Barrantes et al., 2005). This method relies on changing the bytecode randomly and recovering it before execution on the processor. We build on this idea by using diversity to change the format of the MAVLink instruction set so that the same functionalities are present but in a different syntactical form. Our solution is a novel instruction diversity approach since we are diversifying remote instructions in order to limit the access while existing solutions focus on the local machine instructions in order to thwart code injection. In the rest of this section, we present our solution and a basic application of the general idea. Next, we analyse the solution by considering bruteforce and eavesdropping attacks perpetrated by script kiddies and motivated hackers. Finally, we propose a more robust implementation based on our security analysis result.

5.1 General Idea

Our solution aims to hide the required commands for a takeover attack by randomizing command values for each drone-GCS pair. The communication between authentic nodes is still possible as long as we use the same diversified commands for them. If the attacker tries to send the default protocol value, the behavior of the drone will be unexpected. At best the command will be ignored, at worst it will generate anomalous behaviors (ARDUPILOT, 2017b). The anomalous behavior is visible to a user who is tracking and guiding the drone since the drone does not operate as expected. In such scenario, the user must send an emergency land command by which the drone returns to the nest. After sending this command, the drone shall not accept command from any sources since commands may come from an untrusted source.

Instruction in MAVLink protocol is determined using Message ID field and Payload fields, which contain the arguments for the command. We assume Message ID is w bit long and Payload part of the instruction is d bit long. Since Message ID is too small to define a Command instruction, Message ID plus a few more bytes of the payload are used as the Command instruction identifier. For a basic implementation, we consider d to be the length of these Command indicator bytes. d bits of payload convey a command meaning only when Message ID flags a command instruction (Message ID $= X$). We assume M is the set of all the possible Message IDs and C is the set of all the possible arguments in the payload. A message in M is defined by m_{ID} where ID is the actual Message ID value. Accordingly, we define c_{ID}. We define size-of operator using |.|:

$$M = \{m_1, m_2, m_3, \dots, m_{|M|}\}, |M| = 2^w$$
$$C = \{c_1, c_2, c_3, \dots, c_{|C|}\}, |C| = 2^d$$

The set of all instructions is defined by I. Members of I are defined by tuple (m_{ID}, c_{ID}). Command instructions are represented by (m_X, c_{ID}) and ordinary messages are defined by $(m_{ID,} null), ID \neq X$. We assume σ_M is the set of all the possible permutations over M.Accordingly, we define σ_C the set of all the permutations over C. Since σ_C and σ_M are independent, the set of all the possible instruction permutations (σ_I) is all the combinations of a σ_M member and a σ_C member.

$$M_j \in \sigma_M,\ 1 \leq j \leq |\sigma_M|$$
$$C_k \in \sigma_C,\ 1 \leq k \leq |\sigma_C|$$
$$I_{j,k} \in \sigma_I\ s.t.\ I_{j,k} = (M_j, C_k)$$

We define ρ a strong permutation function that randomly picks an element in $I_{j,k}$. The construction of ρ is purely off-line and is executed only once at the drone installation. Several well-known constructs can be used. For example, the Luby-Rackoff construction (Naor and Reingold, 1999) generates a pseudorandom permutation from a pseudorandom function; similarly, the methodology described in (Goldreich, et al., 1986) can be used to construct a pseudorandom function from a strong pseudorandom generator such as BBS (Blum et al., 1986). Even with small w and d, ρ has large space $|\sigma_I|$ to perform randomization:

$$|\sigma_M| = |M|! = 2^w!$$
$$|\sigma_C| = |C|! = 2^d!$$
$$|\sigma_I| = |\sigma_M| * |\sigma_C| = 2^w! \times 2^d!$$

In practice, if $|M|$ is significantly smaller than $|C|$, as it is for MAVLink version 1.0, attackers will first exploit $(m_{i,} null)$ tuples for messages without argument. For this reason, the defense strength lies on the complexity of finding the random C_k. For implementation, we randomly change the instruction values using ρ before compilation and since the instruction values will be used as identifiers, randomizing their value will not harm the application logic. Interoperability in a UAS is possible as long as drone's firmware and GCS are compiled using the

same header file for instructions.

MAVLink is a header-only message marshalling library (GitHub, 2017b). The messages are defined in one xml file. After applying ρ, its output must be fed to an xml file generator to produce the expected xml file by MAVLink. Next, to create the header file, the header generator script must run with the created xml file as the parameter(ARDUPILOT, 2017a). Later, both GCS and the firmware should be compiled using the same message header file. Finally, the firmware code should be flashed into the autopilot. Vendors can write a script that does the following:

1. Generates a new instruction set for each user using ρ
2. Creates message definition XML file
3. Generates the header file
4. Compiles the code
5. Updates the drone firmware and GCS

It is very important for the vendor to provide this solution as an automatic update mechanism that users can regularly run. Since these steps are run before the flight and when the system is offline, the overhead of this solution is zero. This solution does not alter the execution paths, hence during flight time, the operation is the same; the only difference is matching commands with a different vector. Moreover, the additional space required for this solution is zero. Finally, this solution by default is resistant to script kiddies since the off-the-shelf exploits do not work anymore. This solution will work "off-the-shelf" for the current MAVLink implementations with little more than a header-file replacement in contrast to other proposed solutions that require changing the packet structure.

5.2 Security Analysis

Bruteforcing, which is trying all the possible combinations, is the first cyber-attack that adversaries may mount against our defense. In *brute force*, the attacker tries all the possible values for a command to retrieve the actual command value. In order to perform an attack, the attacker needs to know at least r instructions required for that type of attack. b of these instructions belong to the set M and the rest is from the set C. The attacker needs to perform BF number of trials to find r commands:

$$\mathrm{BF} = \sum_{i=0}^{b-1}(|M| - i) + \sum_{j=0}^{r-b}(|C| - j)$$

In case of stealthy takeover on MAVLink, b is 10 and r is 16 (see Table 1). BF for this attack is 395716 trials. Although 395716 operations in cryptography is totally breakable, in our scenario it is strong enough. The complexity of breaking this defense lies on the difficulty of verifying whether the command had the expected effect without human aid. For instance, if the attacker is searching for the command to throttle, she ca0nnot bruteforce all the possible commands in a loop because after transmission of each command she needs to observe physically the drone and see if it had the desired throttle effect.

An alternative to brute force attack against these systems is eavesdroping. The attacker eavesdrops the communication between the drone and the GCS to extract the commands he needs for hijacking. The analysis of the eavesdropped traffic for command extraction can be performed automatically or manually. A combination of the two approaches would form a known plain-text attack. Mainly for automatic *analysis* the attacker has two choices:

- Pattern matching
- Commands frequency analysis

Pattern matching is based on two examinations. The first examination is the length of the payload and the number of parameters. This method, though, is not really effective because any command with the same number of parameters has almost the same format in a communicated packet. This is because the traffic contains value also for empty parameters and autopilot ignores these values (Meier et al., 2011). The second examination is on the domain of the value. This check will also generate many false positives since two commands may have same parameter domains while doing different things. An example of such is MAV_CMD_NAV_LAND and. MAV_CMD_NAV_VTOL_TAKEOFF commands that both have the same number of parameters and domains but do exactly opposite things.

Commands frequency analysis is not conclusive too. By measuring the frequency of some commands, the attacker may find the most common messages. This is possible mainly because the instruction space is limited. Moreover, not all of this space is used; Message IDs 180 – 240 and more than 99% of MAV_CMD commands are unused (Meier et al., 2011). Additionally, some commands are transmitted more frequent than the others are. For example, HEARTBEAT message is transmitted based on an interval and analyzing that can give a hacker the insight about both the interval and the heartbeat command. Waypoint commands are other types of commands that can be detected; however, these commands are not enough for a stealthy takeover. As mentioned in section 2 if the attack is not stealthy, it is unlikely to be successful in a practical scenario.

An alternative to the automatic traffic analysis is manual *behavioral analysis*. A computer may not

distinguish the behavior of a drone after a command; however, a human can do so. We call this approach behavioral analysis since it depends on the human intelligence to detect a command's effect. The attacker may choose *behavioral analysis* by simply observing the drone behavior after capturing a command. This method can also be integrated by the previous method to simplify the attacker's work. Nevertheless, some commands may be revealed by a very low probability essentially because they are not frequent at all. CHANGE_OPERATOR_CONTROL and MAV_CMD_SET_MESSAGE_INTERVAL are such commands that are essential for stealthy takeover of a Radio drone. As presented in Table 1, these commands allow the attacker to disconnect the legitimate user and mislead her. Additionally, if the attacker again chooses to perform an overt attack on a WIFI drone, disregarding these commands, he has to wait for the MAV_CMD_NAV_LAND command to be sent. After sending this command, the attacker has a very limited time to intercept the command and run the attack (less than 1 minute) because after that the drone lands and it will be disarmed. Even the attacker succeeds, the drone is few meters away from the legitimate user.

A motivated attacker may crack the commands listening to airwaves for a long time. Then, the attacker may try to steal the drone during the next flight while disconnecting the owner. Regularly updating the drones' firmware and the GCS mitigates the threat of leaking sensitive commands.

Both in bruteforce attack and eavesdropping, issuing command, while the legitimate owner controls the drone, results in unexpected behavior. The unexpected behavior of the drone will violate the 'stealthy drone takeover' requirements. The jig-jagged erratic behavior of the drone is an indicator of compromise. Based on this, a proper incident response such as security emergency landing can be realized by adding a command that initiates return-to-nest process and disables remote control.

5.3 Enhancement

A more robust solution to the aforementioned attacks would be including also the arguments of the MAVLink command in the random permutation. This would require a compiled bit-shift preprocessor that regenerates the original arguments. A fixed permutation of d bits can be done with $O(d)$ instructions (Lee et al., 2001). This will increase the search space of a bruteforce attack to more than 256 bits (Message ID + Payload), which renders a bruteforce search infeasible in practice. Known plain texts attacks will not be effective as well. A known plain text attack may only succeed based on two premises. Firstly, the attacker must recover the parameters of a command. This requires precision at bit level to identify target GPS coordinates for GPS-oriented commands by looking at the drone position. Secondly, the attacker's infrastructure must be capable of cracking the permutation in a small window of 20 minutes. The script kiddies and motivated attackers cannot fulfil these two premises. That said, the overhead and implementation cost of this solution is not zero. However, in comparison to AES (see Table 3), there is no network traffic overhead for this solution. Moreover, the instruction overhead of this solution is definitely less than SHA and AES; they require far more instructions for encryption than 256 ones that this solution needs to rebuild a payload. Compatibility, Performance and protection comparison of all the security solutions are shown in Table 4.

6 CONCLUSIONS

In this research, we discuss the security weaknesses of drones and how hackers have leveraged these weaknesses in recent years to take control over the drones. We, further, present the concept of stealthy takeover attacks as a practical evolution of state-of-the-art attacks. Despite the current unavailability of security solutions, improvements based on the drone constraints are required; in particular, we argue how an encryption oriented security solution can generate significant overhead on the network and increase energy consumption. Taken into account the threats UAV systems face, we employ Moving Target Defence instruction diversity technique to secure drones. We show how our solution has better performance than existing solutions and offer acceptable defence against script kiddies and the motivated hackers. Even if attackers partially succeed in finding some commands, the incident response is possible because the attack would not be stealthy.

A more robust solution would be Payload bit permutation that would be resilient against the threats we consider (script kiddies and the motivated attackers). Clearly, a powerful funded organization may afford the cost required to satisfy the premises of cracking bit permutation. However, this is out of the scope for a commercial solution addressing the commodity drones.

Table 4: Comparison of MAVLink security solutions.

		GIDL	sMavlink	Packet Signing	Diversity
Compatibility	Protocol alteration	Severe	Low	Moderate	Low
	Backward Compatibility	No	No	Partly	No
	Implementation complexity	Extremely High	High	Moderate	Low
Performance	Computational overhead	Low	Medium	Low	None
	Memory overhead	Low	Low	Low	Very Low
Protection	Confidentiality	Yes	Yes	No	Partly(*)
	Access control	Yes	Yes	Yes	Yes
	DOS defense	No	No	No	No

- (*) requires coupling digital interception with physical observation and reverse kinematics.

ACKNOWLEDGEMENTS

The authors thank the anonymous reviewers, Professor Van de Pol and Redsocks Security researchers for their valuable comments.

REFERENCES

UAV Expert News (2016), "Drone Wars has a new front on Retailers", November, available at: http://www.uavexpertnews.com/drone-wars-has-a-new-front-on-retailers-amazon-prime-air-drone-unveiled/ (accessed 19 December 2016).

Rodday, N., de Oliveira Schmidt, R. and Pras, A. (2016). Exploring Security Vulnerabilities of Unmanned Aerial Vehicles.

Meier, L., Camacho, J., Godbolt, B. and Goppert, J. (2011.). "MAVLINK Common Message Set", MAVLINK Common Message set specifications, available at: https://pixhawk.ethz.ch/mavlink

Google Groups - Toward Secure MAVLink. (2013), available at: https://groups.google.com/d/topic/mavlink/u0UK_-wRVSI (accessed April 10, 2016).

Meier, L., Gho, G., Karapanos, N., & S. (2013, August). SMAVLink Request for Comments, available at: https://docs.google.com/document/d/1upZ_KnEgK3Hk1j0DfSHl9AdKFMoSqkAQVeK8LsngvEU/edit#

Tridgell, A., Meier, L. (2015). MAVLink 2.0 packet signing proposal, available at: https://docs.google.com/document/d/1ETle6qQRcaNWAmpG2wz0oOpFKSF_bcTmYMQvtTGI8ns/edit#heading=h.r1r08t7lr2pc (accessed 7 April 2016).

Pike, L. (2013), "Keynote talk I: Building a high-assurance unpiloted air vehicle", paper presented at Eleventh IEEE/ACM International Conference on Formal Methods and Models for Codesign (MEMOCODE), October, pp. 33-34.

Evans, D., Nguyen-Tuong, A., & Knight, J. (2011), Effectiveness of moving target defenses, Springer New York Columns on Moving Target Defense, pp. 29-48.

Larsen, P., Homescu, A., Brunthaler, S., & Franz, M, (2014), SoK: Automated software diversity, 2014 IEEE Symposium on Security and Privacy (SP), May, pp. 276-291.

Deligne, E. (2012), ARDrone corruption, Journal in Computer Virology, Vol. 8 No.1, pp. 15-27.

Kamkar, S. (2013), SkyJack, available at: https://github.com/samyk/skyjack. (accessed 7 April 2016).

Marty, J. A. (2013), Vulnerability analysis of the mavlink protocol for command and control of unmanned aircraft (No. AFIT-ENG-14-M-50), Air force institute of technology wright-patterson afb oh graduate school of engineering and management.

Layer, M. P. (2013), Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs).

Pike, L., Hickey, P., Bielman, J., Elliott, T., DuBuisson, T., Launchbury, K. (2013.), Secure MAVLink (SMAVLink), available at: http://smaccmpilot.org/software/commsec-overview.html (accessed April 10, 2016).

Anderson, R., & Kuhn, M. (1996), "Tamper resistance-a cautionary note", in Proceedings of the second Usenix workshop on electronic commerce, November, Vol. 2, pp. 1-11.

Spears, R., and R. Melgares. (2011), "ZigBee security: find, fix, finish.", presentation presented at ShmooCon, Washington, D.C.

Sasi, R. (2015, January), Maldrone the first backdoor for drones, Fb1h2s aka Rahul Sasi's Blog.

Fourty, N., Van Den Bossche, A., & Val, T. (2012), "An advanced study of energy consumption in an IEEE 802.15. 4 based network: Everything but the truth on 802.15. 4 node lifetime", Computer Communications, Vol. 35 No.14, pp. 1759-1767.

Potlapally, N. R., Ravi, S., Raghunathan, A., & Jha, N. K. (2003, August), "Analyzing the energy consumption of security protocols", in Proceedings of the 2003 ACM international symposium on Low power electronics and design, pp. 30-35.

Barrantes, E. G., Ackley, D. H., Forrest, S., & Stefanović, D. (2005), "Randomized instruction set emulation", Transactions on Information and System Security (TISSEC), ACM, Vol. 8 No. 1, pp. 3-40.

Wash, R. (2010), “Folk models of home computer security”, in Proceedings of the Sixth Symposium on Usable Privacy and Security, July, p. 11.

Naor, M., & Reingold, O. (1999), “On the Construction of Pseudorandom Permutations: Luby—Rackoff Revisited”, Journal of Cryptology, Vol. 12 No. 1, pp. 29-66.

Goldreich, O., Goldwasser, S., & Micali, S. (1986), “How to construct random functions”, Journal of the ACM (JACM), Vol. 33 No. 4, pp. 792-807.

Blum, L., Blum, M., & Shub, M. (1986), “A simple unpredictable pseudo-random number generator”, SIAM Journal on computing, Vol. 15 No. 2, pp. 364-383.

Lee, R. B., Shi, Z., & Yang, X. (2001), “Efficient permutation instructions for fast software cryptography”, IEEE Micro, Vol. 21 No. 6, pp. 56-69.

GitHub (2017a) https:// github.com/mavlink.

DIGI (2017) http://knowledge.digi.com/articles/ Knowledge_Base_Article/Maximum-payload-size-for-Digi-RF-products.

ARDUPILOT (2017a) http://ardupilot.org/dev/docs/ code-overview-adding-a-new-mavlink-message.html.

Shellntel (2017) http://www.shellntel.com/blog/2015/ 9/25/drone-code-execution.

ARDUPILOT (2017b) http://ardupilot.org/plane/docs/ common-mavlink-mission-command-messages-mav_cmd.html#common-mavlink-mission-command-messages-mav-cmd.

GitHub (2017b) https://github.com/mavlink/c_library/ blob/master/message_definitions/common.xml.