HoneyDrone: a Medium-Interaction Unmanned Aerial Vehicle Honeypot

Jörg Daubert*, Dhanasekar Boopalan†, Max Mühlhäuser*, Emmanouil Vasilomanolakis*
Telecooperation Lab,
Technische Universität Darmstadt
Darmstadt, Germany
*{daubert, max, vasilomano}@tk.tu-darmstadt.de
†dhanasekar.boopalan@stud.tu-darmstadt.de

***Abstract*—Over the last years, we have experienced an increased utilization of Unmanned Aerial Vehicles (UAVs) not only in personal, but also commercial and public safety applications. Simultaneously, malicious activities have emerged too; from hijacking of UAVs (and their cargo), to the theft of private information stored in UAVs, attacks not only exist but seem to increase both in their numbers and their sophistication. In this paper, we propose *HoneyDrone*, the first honeypot that is specifically designed for the protection of UAVs. *HoneyDrone* emulates a number of UAV-specific and UAV-tailored protocols, making it possible to lure adversaries into attacking it. The honeypot is designed to run in portable low-cost devices, e.g., Raspberry Pis, which makes it possible to strategically deploy it in a variety of locations. Our system can assist in detecting active attackers in a certain area, as well as in shedding light into the adversaries' techniques for compromising UAVs. We evaluate *HoneyDrone's* performance and also examine a number of different realistic attack scenarios to show how the honeypot can cope with them.**

I. Introduction

Unmanned Aerial Vehicles (UAVs), commonly referred to as *drones*, have gained massive popularity in recent years. UAVs are used for personal purposes, e.g., taking pictures and videos, as well as professional applications, such as agricultural surveys and package deliveries. Furthermore, even mission critical operations including medication delivery services and health and safety inspections are performed with UAVs nowadays. As UAVs are computer-controlled systems with radio/wireless interfaces, attacks and attack scenarios against these systems are surfacing [8], [11], [12]. For instance, these may include wiretapping and theft of data, mission interference, and even the theft or misuse of UAVs. To cope with this attack landscape, we propose the idea of a portable drone honeypot; a security mechanism which can emulate the protocols that are utilized by UAVs and lure attackers into it.

A honeypot is a system whose only value lies in being probed, attacked, and/or compromised [6]. In detail, such systems have no real production value, but instead they appear to be vulnerable and thus attractive to attackers. Traditionally, honeypots are utilized as early warning defense mechanisms, as methods for studying adversaries and their techniques, as well as a way to reduce the attack surface of the monitored network [6].

On top of the aforementioned functionalities of honeypots, we argue that due to certain properties of the UAVs (namely the signal strength property and the ability of the UAV to quickly traverse an area) a drone honeypot introduces additional benefits. In particular, in a drone-attack scenario the adversary does not have to maintain visual of the target, but instead can rely on their signal strength for attacking and hijacking the drone; for example, by utilizing a high-gain antenna. Therefore, we argue that a UAV honeypot is able not only to detect a drone attack but even mitigate it as long as: (i) it has a stronger signal than the actual drone (which for example can be achieved with proper antennas) and (ii) that is placed in a strategic location. This scenario is also illustrated in Figure 1.

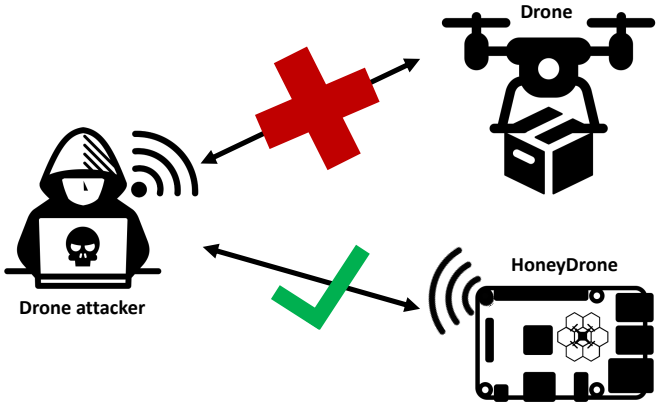

Fig. 1. High level view of the honeypot usage scenario

In this paper, we propose *HoneyDrone*, a novel *medium-interaction portable drone honeypot*. *HoneyDrone* provides a medium-interaction interface for many UAV-specific and UAV-tailored protocols. This allows for the emulation, recording and analysis of malicious activity in UAVs. *HoneyDrone* is also portable, in the sense that is lightweight and can be deployed in low-cost small devices, e.g., a Raspberry Pi. This portability property is important as it makes it possible to dynamically deploy the honeypot in various locations. In addition, we argue that with our honeypot it is possible to

guide attackers away from real UAVs and delay them, whilst reducing the overall attack surface of the monitored area. Finally, to the best of our knowledge this is the first honeypot which is specifically designed for UAVs and UAV protocols.

The rest of this paper is structured as follows. In Section II, we discuss the related work in the fields of honeypots. Furthermore, Section III introduces *HoneyDrone* with an emphasis on its architecture and the particular modules that constitute it. Subsequently, Section IV, evaluates the proposed system in terms of its performance as well as its ability to cope with realistic attack scenarios. Lastly, Section V concludes this paper and gives some insights on our future work plans.

II. Related Work

Honeypots are systems whose only value lies in being probed, attacked, and/or compromised [6]. Having no real production value, any communication or interaction with such a system is considered an attack. Honeypots can be classified to *low*-, *medium*- and *high*-interaction with respect to the level of interaction they offer to the adversary.

On the one hand, *high*-interaction honeypots are real systems, e.g., a Virtual Machine (VM), that exhibit certain vulnerabilities and are closely monitored. These systems, however, are very expensive to maintain and have the risk of being compromised. On the other hand, *low*- and *medium*-interaction honeypots only *emulate* protocols, in a different granularity (low and hight respectively). Such systems are nowadays the preferred option, compared to *high*-interaction ones, for several reasons [9]. First, they are much cheaper to maintain as they can offer explicit and detailed logging and monitoring functionalities. Moreover, it is easier to develop secure and contained *low*- and *medium*-interaction honeypots.

Name	**Generic Protocols**	**MAVLink Protocol**	**FS**	**Config**	**Drone Radio**
Heralding [2]	FTP,SSH Telnet	✗	✗	✗	✗
Kojoney2 [4]	SSH	✗	✗	✗	✗
Cowrie [7]	SSH Telnet	✗	✗	✓	✗
HosTaGe [16]	FTP,SSH Telnet	✗	✗	✓	✗
HoneyPy [3]	TCP,UDP	✗	✗	✓	✗
HoneyWRT [1]	Telnet	✗	✗	✓	✗
Bluepot [13]	L2CAP RFCOMM OBEX	✗	✗	✗	BT

TABLE I
Analysis of honeypot drone capabilities in the state of the art

As summarized in Table I, a number of state-of-the-art honeypots [5], [6], [10] can monitor and emulate several general purpose protocols, such as Telnet, SSH, and FTP. Nevertheless, to the best of our knowledge, there seems to be no honeypot supporting the drone-specific MAVLink protocol. Likewise, none of the honeypots seems to be able to take an extracted File System (FS) from a drone, emulate the FS, and record the changes. Similarly, most of the honeypots are designed for one fixed use case and lag the option to be configured to emulate different devices easily. Only a few honeypots have been designed to be mobile or portable [14], [17], [18], offering the ability to be placed close to the operating area of UAVs or perhaps even mounted to a UAV directly. Finally, it is vital for a honeypot to emulate a drone's radio interfaces; none of the honeypots is designed to emulate and log WiFi radios, only Bluepot [13] can emulate the (rarely used) Bluetooth radio.

III. HoneyDrone Architecture

This section proves a brief background in UAV communication, explains how our honeypot leverages these communication capabilities, and presents its architecture.

A. UAV Communication

Many of today's UAVs often use radio bands and protocols that are widely supported by commodity hardware and software defined radios. Even though this fact makes attacks on UAVs affordable on the one hand, it also allows us to construct the low-cost *HoneyDrone* on the other hand. Examples are budget UAVs with Wi-Fi and Bluetooth for command and control. Expensive and professional UAVs build on vendor-specific radio protocols, e.g., Lightbridge[1] and SiK Radio[2]. Nevertheless, these protocols operate frequency bands with a high availability radio interfaces and software-defined radios.

On top of the radio communication, UAVs use application protocols that have been attacked in the past: Telnet, SSH, FTP, and more recently MAVLink[3] [15]. Again, these protocols are easy to attack, but are also well-defined enough to be implemented within a honeypot.

Our portable *HoneyDrone* leverages these properties to emulate drone radio interfaces on cheap commodity hardware, and by offering low to medium interaction and emulation for many of the aforesaid communication protocols. In addition, the *HoneyDrone* emulates all relevant properties for a range of commercial and self-build UAVs for command and control.

B. HoneyDrone *Architecture*

Our first generation portable drone honeypot is designed around cheap mini computers, such as the Raspberry Pi family. The supported features include the radio protocols Wi-Fi with the application protocols Telnet, SSH, FTP, and MAVLink. These features are well suited for consumer UAVs like the AR Drone; even professional self-build UAV use this protocol stack for MAVLink telemetry.

In order for a honeypot the emulate UAVs, several very specific challenges have to be resolved: first, UAVs use vendor-specific Wi-Fi properties, such as ESSID, BSSID, authentication, encryption, IP assignment, etc. Unlike existing honeypots, *HoneyDrone* emulates such properties. Second, UAVs are physical objects moving in the real world, which reflects into (MAVLink) telemetry and flight recorders. The *HoneyDrone*, therefore, uses a drone simulator to generate realistic telemetry and input responses.

[1] www.dji.com/lightbridge-2
[2] github.com/ArduPilot/SiK
[3] diydrones.com/profiles/blogs/hijacking-quadcopters-with-a-mavlink-exploit

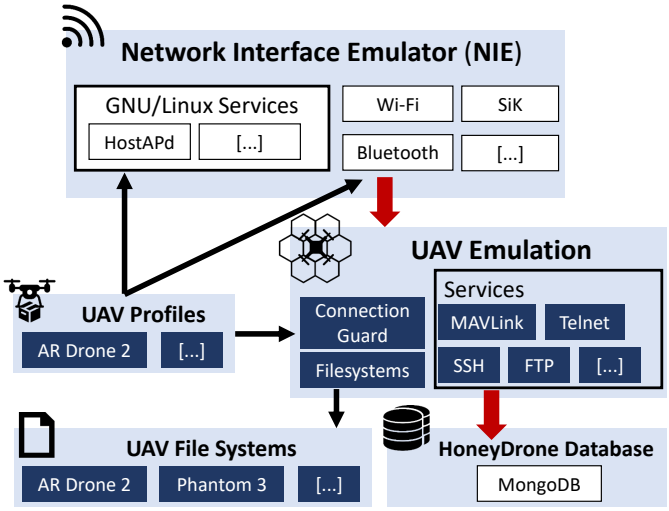

Fig. 2. HoneyDrone architecture

At a glance, the honeypot prototype is using a Raspberry Pi, running Raspbian, in combination with an Alfa AWU036NH wireless adapter, and the Hostapd and Dnsmasq. The software core is implemented in Python 2.7 with the Twisted framework. A combination of PyMAVLink, MAVProxy and the Ardupilot Software In The Loop (SITL) simulates drones with MAVLink access. The *HoneyDrone* takes UAV profiles from a configuration file and sets up all components to emulate a UAV down to Wi-Fi MAC addresses and UAV filesystems (contents extracted from real drones). Lastly, all attacker input is recorded to a MongoDB database.

In detail, the *HoneyDrone* is comprised of the components as depicted in Figure 2:

1) **Network Interface Emulator (NIE)**: The NIE is responsible for setting up the communication/network interface of the *HoneyDrone* host. The network interfaces supported by this architecture are Wi-Fi, Bluetooth, SiK Radio (serial interface). Upon start, the NIE takes the interface settings for a particular UAV from the configuration file, checks for the availability of network devices and brings up the interface. These settings include the ESSID and BSSID of a UAV—the device parts of the BSSID can be specific or autogenerated as desired. Next, additional services, such as HostAPd, are configured and launched to emulate for instance the authentication of a particular UAV. The configuration commands as well as the daemon log files are written to the database.
2) **UAV Emulation**: The UAV Emulation is the core of *HoneyDrone*. The emulation handles incoming connections, creates objects of specific protocols, e.g., Telnet, and continuously monitors and logs the traffic. The emulation takes its input from the configuration file, similarly to the NIE, to customize and emulate specific protocols on a network interface. The customization is crucial as UAVs protocol implementations differ; for Telnet and SSH that includes the prompt, the available commands, and the whole FS as such. All these functionalities are realized by three sub-components:
   - **Services**: The services are a set of classes or modules, each for a different protocol. In more details, a service defines a set of variables and methods to emulate a particular protocol on the *HoneyDrone*. The current *HoneyDrone* prototype fully supports Telnet, SSH, FTP, and MAVLink for a set of UAVs. To the best of our knowledge, the *HoneyDrone* is the first honeypot supporting MAVLink. For that we use the Ardupilot SITL[4] simulator in combination with MAVProxy and PyMAVLink. The SITL simulates one of the most popular flight control firmwares, the Ardupilot family, which includes for instance the Arducopter. MAVLink relays the communication from a network interface to a SITL instance, while allowing to log all commands, responses, and telemetry. PyMAVLink converts the MAVLink message to a human-readable form before recording them into the database.
   - **Connection Guard**: While the *HoneyDrone* is more "physical" than existing honeypots due to the radio interfaces, it is designed to handle multiple attackers simultaneously. The Connection Guard is responsible for spawning services, limiting the number of connections if required, and for detecting and recovering services' failures.
   - **File System Emulator**: UAVs differ significantly in their FSs. We, therefore, separated the FS structure from the emulation. The emulator loads an FS from the Drone FS (see below) to match a particular UAV. All modification performed by an attacker are recorded in a transient overlay and logged to the database.
3) **Drone File System**: The Drone FS refers to the emulated FS of a specific drone. For instance, some UAVs expose the Linux $/bin$ folder or a folder with flight recordings, while others do not. To correctly emulate these differences, we extract the FSs from real UAVs and convert them to a file. Files uploaded by an attacker are persisted and logged for later investigation.
4) ***HoneyDrone* Database**: The database component collects and stores all the information in MongoDB. We selected this database for its excellent RESTful/JSON export capabilities and the document-oriented structure. The latter feature allows for fast and easy addition, extension and modification of stored record types. In the current prototype, the database will, for instance, record connection (attempts), (invalid) credentials, and issues commands (valid and invalid) for the supported application protocols.
5) **Configuration file**: This component refers to a single or a set of configuration files that are used by the NIE at the time of network interface setup and by the

[4]http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

UAV Emulation while the *HoneyDrone* is active. The configuration contains properties such as the type of network interface for the emulated drone, the FS to be chosen, and the settings for the protocols to be emulated. We are using an INI-style format for the configuration. This format is compact and human-readable to allow for quick and easy adjustments; for instance to alter the emulated ESSID to match a real UAV.

In summary, while several existing GNU/Linux and open source components are used by the *HoneyDrone*, the correct integration, configuration, execution, monitoring, and data extraction is unlike any other honeypot. Moreover, the tight coupling of *HoneyDrone* with physical properties, e.g., the wireless network interfaces and the UAV simulation, significantly distinguishes the architecture from other low-interaction honeypots.

IV. Evaluation

The evaluation of the *HoneyDrone* prototype is twofold. First, we study its performance by measuring the CPU utilization in different scenarios and while running *HoneyDrone* in a Raspberry Pi. Second, we emulate two attack scenarios, focusing on the Telnet and MAVLink protocols respectively.

A. Performance Evaluation

The *HoneyDrone* is designed to be deployed close to real drone locations and potential attackers. Therefore, the *HoneyDrone* is only required to handle attackers within physical/wireless range as opposed to honeypots running on an Internet scale. Still, we aim at high efficiency to be able to run *HoneyDrone* on low-power and even battery-powered devices. The ultimate goal is to carry *HoneyDrone* with real drones or to run *HoneyDrone* as part of a drones flight controller. For that, we evaluate the honeypot's CPU utilization on a Raspberry Pi 2 model B v1.1 with a 900MHz ARM Cortex-A7 quad-core CPU and 1GB of RAM.

We measured the CPU utilization in multiple stages starting with an idle Raspbian system, incrementally enabling *HoneyDrone* services, connecting simulated attackers, and finally letting the attackers interact with the honeypot.

During the *HoneyDrone* software startup, the CPU utilization reaches its maximum with 48% over the idle Raspbian system. With all current services—including the physical simulation of a drone—running, the CPU utilization reaches an average 18% above idle. Here, the drone and MAVLink protocol simulation account for the vast majority of this utilization. Each adversarial connection to a service, such as Telnet/SSH/MAVLink, incurs up to another 2% of CPU utilization. Therefore, the *HoneyDrone* can support multiple parallel connections without any significant overhead in its performance on a low-power portable computer.

B. Attack Scenarios

In the following, we emulate two attack scenarios (a Telnet and a MAVLink attack) and show how *HoneyDrone* deals with them.

1) Telnet attack: In this attack scenario, we perform a Telnet attack on a real *Parrot AR Drone 2.0* followed by a similar attack on the *HoneyDrone*. In order to hijack the drone, it is necessary to de-authenticate the device which is currently controlling the drone. When such de-authentication attack is carried out on clients connected to the *HoneyDrone*, it neither causes any damage to the *HoneyDrone* application nor to any hardware.

```
"_id" : ObjectId("5131d0100fa4c416dc63c210"),
"clientinfo" : {
"ip" : "192.168.1.31",
"type" : "TCP",
"port" : 40784},
"eventinfo" : [
{
"timestamp" : "2017-12-20 21:03:12.141377",
"event" : "New Connection "},
{
"timestamp" : "2017-12-20 21:03:15.718012",
"event" : "Command found: `ls -l`"},
{
"timestamp" "2017-12-20 21:04:35.567345",
"event" : "Command found: `rm -r bin`"}
]}
```

Fig. 3. MongoDB log of the Telnet attack

For a Telnet connection to the drone, however, a de-authentication is even not necessary. Hence, it is possible to connect to the drone and issue commands to it even when the drone is mid-flight, thereby shutting down the drone or deleting important files from the drone's operating system. The Telnet connection to the Parrot drone is not password protected and any connected client is granted the root privilege on the FS. In our scenario, a Telnet connection has been established with the drone after connecting to the drone's Wi-Fi network. Once connected, the command "*ls -l*" is issued, to list the contents of the directory. Figure 4, depicts the output of the list command issued on the root directory of the Parrot drone. Figure 4, also shows that the attacker has write privileges on every file; this allows, for instance, to alter and/or delete these files.

The same attack scenario is repeated on the *HoneyDrone* which is configured to run on the same IP address, as used by the Parrot Drone and an open Wi-Fi access point has been set up. Upon establishing a Telnet connection to the *HoneyDrone*, the same listing command is issued in the root directory. In addition, a second command ("*rm -r bin*") is issued to delete the bin folder completely from the root directory. Figure 4, shows the results of the aforesaid commands issued to the *HoneyDrone*.

From Figure 4, it can be seen that the FS of the Parrot drone and the *HoneyDrone* are almost indistinguishable. Furthermore, the deletion of the folder does not cause any harm to the *HoneyDrone* as the FS is not real. *HoneyDrone* logs all the information about the attack in its database. In the scenario discussed above, the honeypot records information about the attacker. In particular, the IP address, the type of connection and the ports used by the attacker as well as the issued commands along with their timestamps. Figure 3,

```
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

BusyBox v1.14.0 () built-in shell (ash)
Enter 'help' for a list of built-in commands.

# ls -l
drwxrwxr-x    4 root     root          5688 Jan  1  1970 bin
drwxr-xr-x    4 root     root          1016 Jan  1  2000 data
drwxrwxrwt    4 root     root          3660 Jan  1  2000 dev
drwxrwxr-x    3 root     root          1256 Jan  1  1970 etc
drwxr-xr-x    2 root     root          1064 Jan  1  2000 factory
drwxr-xr-x    3 root     root           536 Jan  1  2000 firmware
drwxrwxr-x    3 root     root           224 Jan  1  1970 home
drwxr-xr-x    5 root     root          2880 Jan  1  1970 lib
drwxrwxr-x    2 root     root           240 Jan  1  1970 licenses
drwxrwxr-x    2 root     root           160 Jan  1  1970 mnt
dr-xr-xr-x   77 root     root             0 Jan  1  1970 proc
drwxrwxr-x    2 root     root           160 Jan  1  1970 root
drwxrwxr-x    2 root     root          2752 Jan  1  1970 sbin
drwxr-xr-x   12 root     root             0 Jan  1  1970 sys
drwxrwxrwt    3 root     root           200 Jan  1  2000 tmp
drwxr-xr-x    2 root     root           232 Jan  1  2000 update
drwxrwxr-x    8 root     root           544 Jan  1  1970 usr
drwxrwxr-x    2 root     root           352 Jan  1  1970 var
#
```

```
Trying 192.168.1.1...
Connected to 192.168.1.1.
Escape character is '^]'.

BusyBox v1.14.0 () built-in shell (ash)
Enter "help" for a list of built-in commands

# ls -l
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 bin
drwxr-xr-x 1 root root 4096 2017-12-19 16:03 data
drwxr-xr-x 1 root root 4096 2017-12-19 16:01 dev
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 etc
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 factory
drwxr-xr-x 1 root root 4096 2017-12-19 16:05 firmware
drwxr-xr-x 1 root root 4096 2017-12-19 16:01 home
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 lib
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 licenses
drwxr-xr-x 1 root root 4096 2017-12-19 16:01 mnt
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 proc
drwxr-xr-x 1 root root 4096 2017-12-19 16:02 root
drwxr-xr-x 1 root root 4096 2017-12-19 16:05 sbin
drwxr-xr-x 1 root root 4096 2017-12-19 16:01 sys
drwxr-xr-x 1 root root 4096 2017-12-19 16:03 tmp
drwxr-xr-x 1 root root 4096 2017-12-19 16:02 update
drwxr-xr-x 1 root root 4096 2017-12-19 16:01 usr
drwxr-xr-x 1 root root 4096 2017-12-19 16:04 var
# rm -r bin
Deleted /bin
#
```

Fig. 4. Telnet attack scenario: i) *Successful Telnet connection to Parrot AR Drone 2.0* (left) ii) *Successful Telnet connection to HoneyDrone* (right)

shows the logs inserted in to the database. There are three events recorded, one for the new connection establishment and two events for the commands issued by the attacker. Note that when the issued command is not implemented by the *HoneyDrone*, the attacker is shown a default error message (”*Command not found*”) and the issued command is logged in the database.

2) MAVLink attack: The second scenario is an attack using the MAVLink protocol. Such an attack can be carried out on any drone with a weakly secured access point and with the support for MAVLink communication. Using a Ground Control Station (GCS) application that supports MAVLink (namely QGroundControl[5]), the attacker can issue new mission way-points to the drone and steer the drone away from its owner. In this scenario, it is assumed that the drone’s Wi-Fi network has been compromised and an attacker can connect the drone’s Wi-Fi network after de-authenticating the original controller.

Upon connecting to the *HoneyDrone’s* Wi-Fi network, the adversary establishes a connection to the honeypot’s MAVLink service through the UDP port 14550 using the GCS application. Since the *HoneyDrone* runs the SITL simulator, the QGroundControl application gets the simulated parameters such as pitch, GPS coordinates, speed, etc., from the *HoneyDrone*, just like it would receive from a real drone. From the QGroundControl application, telemetry commands have been issued to the drone using the MAVLink protocol. Figure 5, shows the QGroundControl application connected to the HoneyDrone. It can be seen that the QGroundControl has received the new way-point co-ordinates from the attacker and started its mission along the received path. From the adversary’s perspective, the drone appears to fly over a fake location

[5]qgroundcontrol.com

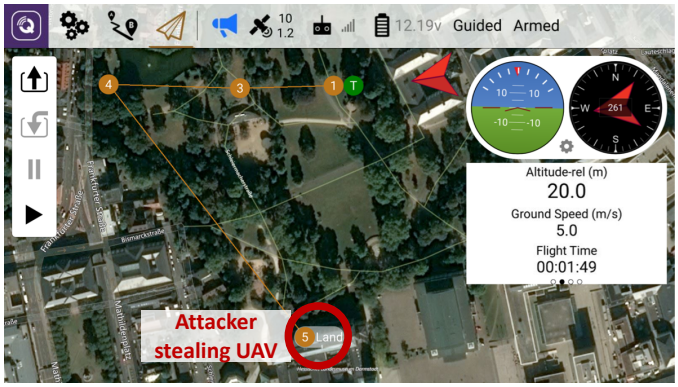

Fig. 5. HoneyDrone QGroundControl connection via MAVLink

as specified in the configuration file of the *HoneyDrone*. The application behaves in the same way as it would behave when it is connected to a real drone.

In our evaluation, new mission tasks, e.g., way-points, have been created and uploaded as a mission to the HoneyDrone (also highlighted in Figure 5). Once uploaded, the *arm* and *takeoff* commands have been sent to the *HoneyDrone*. The honeypot responds by sending the fake GPS coordinates as per the uploaded mission tasks. The QGroundControl software shows the movement of the drone using the parameters received from HoneyDrone. All the MAVLink messages exchanged between the GCS and the *HoneyDrone* are logged into the MongoDB database. Lastly, Figure 6, presents a subset of the logging, containing the GPS coordinates exchanged between the honeypot and the QGroundControl, as saved in the DB of the honeypot.

```
{"timestamp" : "2017-12-20 21:54:06.900751",
  "event" : "MISSION_ITEM {
    target_system : 1,  target_component : 190,
    seq : 0,  frame : 0,  command : 16,
    current : 1,  autocontinue : 1,
    param1 : 0.0,  param2 : 0.0,  param3 : 0.0,
    param4 : 0.0, x : 49.8767967224,
    y : 8.65258693695,  z : 50.0
  }"
}, {"timestamp" : "2017-12-20 21:54:07.022121",
  "event" : "MISSION_ITEM {
  target_system : 1, target_component : 190,
  seq : 1,  frame : 3,  command : 21,
  current : 0,  autocontinue : 1,
  param1 : 0.0,  param2 : 0.0,  param3 : 0.0,
  param4 : 0.0,  x : 49.8753738403,
  y : 8.65268516541,  z : 0.0
}"
```


Fig. 6. MongoDB log for MAVLink attack

V. Conclusion

In this paper, we propose *HoneyDrone*, a honeypot for the detection of attacks in the UAV ecosystem. *HoneyDrone* is the first honeypot that emphasizes solely on drones and their protocols. In addition, it is lightweight in the sense that it can be deployed in small low-cost devices and, hence, can be easily deployed in various locations. We have studied the performance of the honeypot and presented a number of experiments that show the ability of *HoneyDrone* to cope with realistic attack scenarios.

With respect to future work, we plan to further improve *HoneyDrone* by focusing on the improvement of protocols' emulation as well as on tackling a number of limitations that we have identified. For instance, one challenge that we plan to study is the ability of the honeypot to emulate changes in its signal power. In a real-world scenario, an attacker who is connected to a drone would expect a continuously changing behavior in the Wi-Fi connectivity, due to the fact that the drone is usually in movement; this is not the case with the current implementation of *HoneyDrone*. That is, in the current state and considering a static placement of the honeypot the adversary might be able to identify that *HoneyDrone* is a honeypot by examining the signal power of the drone. Finally, we also plan to further evaluate the performance as well as the ability of the honeypot to handle a large amount of connections simultaneously.

VI. Acknowledgments

The research leading to these results has received funding from the European Union's Horizon 2020 Research and Innovation Program, TAKEDOWN, under Grant Agreement No 700688, as well as by the DFG as part of project D.4 within the RTG 2050 "Privacy and Trust for Mobile Users".

References

[1] Honeywrt honeypot. https://github.com/CanadianJeff/honeywrt, 2015.

[2] Heralding honeypot. https://github.com/johnnykv/heralding, 2016.

[3] Honeypy honeypot. https://github.com/foospidy/HoneyPy, 2017.

[4] Justin C. Klein Keane. Kojoney2 honeypot. https://github.com/madirish/kojoney2.

[5] Yin Pa Minn, Shogo Suzuki, Katsunari Yoshioka, Tsutomu Matsumoto, and Christian Rossow. IoTPOT : Analysing the Rise of IoT Compromises. In *9th USENIX Workshop on Offensive Technologies (WOOT)*. USENIX Association, 2015.

[6] Marcin Nawrocki, Matthias Wählisch, Thomas C Schmidt, Christian Keil, and Jochen Schönfelder. A survey on honeypot software and data analysis. *arXiv preprint arXiv:1608.06249*, 2016.

[7] Michel Oosterhof. Cowrie honeypot. https://github.com/micheloosterhof/cowrie, 2014.

[8] Johann-Sebastian Pleban, Ricardo Band, and Reiner Creutzburg. Hacking and securing the ar. drone 2.0 quadcopter: investigations for improving the security of a toy. In *IS&T/SPIE Electronic Imaging*, pages 90300L–90300L. International Society for Optics and Photonics, 2014.

[9] Niels Provos and Thorsten Holz. *Virtual honeypots: from botnet tracking to intrusion detection*. Addison-Wesley Professional, 2007.

[10] Lukas Rist, Daniel Haslinger, John Smith, Johny Vestergaard, and Andrea Pasquale. Conpot honeypot, 2013.

[11] Nils Miro Rodday, Ricardo de O Schmidt, and Aiko Pras. Exploring security vulnerabilities of unmanned aerial vehicles. In *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*, pages 993–994. IEEE, 2016.

[12] Hichem Sedjelmaci, Sidi Mohammed Senouci, and Nirwan Ansari. Intrusion detection and ejection framework against lethal attacks in uav-aided networks: A bayesian game-theoretic methodology. *IEEE Transactions on Intelligent Transportation Systems*, 18(5):1143–1153, 2017.

[13] Andrew Smith. Bluepot: Bluetooth honeypot. https://github.com/andrewmichaelsmith/bluepot, 2013.

[14] Emmanouil Vasilomanolakis, Shankar Karuppayah, Mathias Fischer, Max Mühlhäuser, Mihai Plasoianu, Lars Pandikow, and Wulf Pfeiffer. This network is infected: Hostage-a low-interaction honeypot for mobile devices. In *Proceedings of the Third ACM workshop on Security and privacy in smartphones & mobile devices*, pages 43–48. ACM, 2013.

[15] Emmanouil Vasilomanolakis, Shankar Karuppayah, Panayotis Kikiras, and Max Mühlhäuser. A honeypot-driven cyber incident monitor: lessons learned and steps ahead. In *International Conference on Security of Information and Networks*, pages 158–164. ACM, 2015.

[16] Emmanouil Vasilomanolakis, Shankar Karuppayah, Max Mühlhäuser, and Mathias Fischer. Hostage: a mobile honeypot for collaborative defense. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 330. ACM, 2014.

[17] Emmanouil Vasilomanolakis, Shreyas Srinivasa, Carlos Garcia Cordero, and Max Mühlhäuser. Multi-stage attack detection and signature generation with ics honeypots. In *NOMS*, pages 1227–1232, 2016.

[18] Matthias Wählisch, Sebastian Trapp, Christian Keil, Jochen Schönfelder, Jochen Schiller, et al. First insights from a mobile honeypot. In *Proceedings of the ACM SIGCOMM 2012 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 305–306. ACM, 2012.