实验结果表面，模糊测试技术和BMC技术可以

Finding Security Vulnerabilities in Unmanned Aerial Vehicles Using Software Verification

Omar M. Alhawi
University of Manchester
Manchester, UK
omar.alhawi@manchester.ac.uk

Mustafa A. Mustafa
University of Manchester, UK &
imec-COSIC, KU Leuven, Belgium
mustafa.mustafa@manchester.ac.uk

Lucas C. Cordiro
University of Manchester
Manchester, UK
lucas.cordeiro@manchester.ac.uk

模糊测试+边界模型检查（BMC）+通信链路安全

***Abstract*—The proliferation of Unmanned Aerial Vehicles (UAVs) embedded with vulnerable monolithic software has recently raised serious concerns about their security due to concurrency aspects and fragile communication links. However, verifying security in UAV software based on traditional testing remains an open challenge mainly due to scalability and deployment issues. Here we investigate software verification techniques to detect security vulnerabilities in typical UAVs. In particular, we investigate existing software analyzers and verifiers, which implement fuzzing and bounded model checking (BMC) techniques, to detect memory safety and concurrency errors. We also investigate fragility aspects related to the UAV communication link. All UAV components (e.g., position, velocity, and attitude control) heavily depend on the communication link. Our preliminary results show that fuzzing and BMC techniques can detect various software vulnerabilities, which are of particular interest to ensure security in UAVs. We were able to perform successful cyber-attacks via penetration testing against the UAV both connection and software system. As a result, we demonstrate real cyber-threats with the possibility of exploiting further security vulnerabilities in real-world UAV software in the foreseeable future.**

***Keywords*-UAV; Software Verification and Testing; Security;**

I. Introduction

Unmanned Aerial Vehicles (UAVs), also sometimes referred to as *drones*, are aircrafts without human pilots on board; they are typically controlled remotely and autonomously and have been applied to different domains (e.g., industrial, military, and education). In 2018, PWC estimated the impact of UAVs on the UK economy, highlighting that they are becoming essential devices in various aspects of life and work in the UK. The application of UAVs to different domains are leading to GBP 42bn increase in the UK's gross domestic product and $628,000$ jobs in its economy [1].

With this ever-growing interest also comes an increasing danger of cyber-attacks, which can pose high safety risks to large airplanes and ground installations, as witnessed at the Gatwick airport in the UK in late 2018, when unknown UAVs flying close to the runways caused disruption and cancellation of hundreds of flights due to safety concerns [2]. Recent studies conducted by the Civil Aviation Authority show that a 2kg UAV can cause a critical damage to a passenger jet windscreen [3]. Therefore, it remains an open question whether the *Confidentiality*, *Integrity*, and *Availability* (CIA) triad principles, which is a model designed to guide policies for information security [4], will be maintained during UAVs software development life-cycle.

UAVs typically demand high-quality software to meet their target system's requirements. Any failures in embedded (critical) software, such as those embedded in avionics, might lead to catastrophic consequences in the real world. As a result, software testing and verification techniques are essential ingredients for developing systems with high *dependability* and *reliability* requirements, needed to guarantee both user requirements and system behavior.

Bounded Model Checking (BMC) was introduced nearly two decades ago as a verification technique to refute safety properties in hardware [5]. However, BMC has only relatively recently been made practical, as a result of significant advances in Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT) [5]. Nonetheless, the impact of this technique is still limited in practice, due to the current *size* (e.g., number of lines of source code) and *complexity* (e.g., loops and recursions) of software systems. For instance, when a BMC-based verifier symbolically executes a program, it encodes all its possible execution paths into one single SMT formula, which results in a large number of constraints that need to be checked. Although BMC techniques are effective in refuting properties, they still suffer from the state-space explosion problem [6].

Fuzzing is a successful testing technique that can create a substantial amount of random data to discover security vulnerabilities in real-world software [7]. However, subtle bugs in UAVs might still go unnoticed due to the large state-space exploration, as recently reported by Chaves et al. [8]. Additionally, according to Alhawi et al. [9], fuzzing could take a significant amount of time and effort to be completed during the testing phase of the software development life-cycle in addition to its code coverage issues. Apart from these limitations, fuzzing and BMC can enable a wide range of verification techniques. Some examples include automatic detection of bugs and security vulnerabilities, recovery of corrupt documents, patch generation, and automatic debugging. These techniques have been industrially adopted

by large companies, including but not limited to Amazon Web Service (CBMC [10]), Microsoft (SAGE [11]), IBM (Apollo [12]), and NASA (Symbolic PathFinder [13]). For example, the SAGE fuzzer has already discovered more than 30 new bugs in large shipped Windows applications [11]. Nonetheless, an open research question consists of whether these techniques can be useful in terms of correctness and performance to verify UAV applications.

Our research investigates both fuzzing and BMC techniques to detect security vulnerabilities in real-world UAV software automatically. Thus, our main research goal is to allow the development of software systems, which are immune to cyber-attacks and thus ultimately improve software reliability. According to the current cyber-attacks profile concerning advanced UAVs, it becomes clear that the current civilian UAVs in the market are insecure even from simple cyber-attacks. To show this point of view, we highlight in our study real cyber-threats of UAVs by performing successful cyber-attacks against different UAV models. These cyber-attacks led to gain a full unauthorized control or cause the UAVs to crash. We show that pre-knowledge of the receptiveness of the UAV system components is all what attackers need to know during their reconnaissance phase before exploiting UAV weaknesses.

A. Contributions

Our main contribution is to propose a novel approach for detecting and exploiting security vulnerabilities in UAVs. We leverage the benefit of using both fuzzing and BMC techniques to detect security vulnerabilities hidden deep in the software state-space. In particular, we make three significant contributions:

- Provide a novel verification approach that combines fuzzing and BMC techniques to detect software vulnerabilities in UAV software.
- Identify different security vulnerabilities that UAVs can be susceptible to. We perform real cyber-attacks against different UAV models to highlight their cyber-threats.
- Evaluate a preliminary verification approach called "UAV fuzzer" to be compatible with the type of UAV software used in industry to exploit their vulnerabilities.

Although our current work represents an ongoing research, preliminary results show that fuzzing and BMC techniques can detect various software vulnerabilities, which are of particular interest to UAV security. We are also able to perform successful cyber-attacks via penetration testing against the UAV both connection and software system. As a result, we demonstrate real cyber-threats with the possibility of exploiting further security vulnerabilities in real-world UAV software in the foreseeable future.

B. Organisation

The rest of the paper is organised as follows. Section II describes the UAVs structure and the recent cyber attacks, in addition to other approaches used to verify security in UAVs. Section III introduces UAV software, UAV communication layer and the methodology to verify it using Fuzzing and Bounded Model Checking techniques. Section IV then describes the benchmarks used and present the results to determine the effectiveness of our approach. Finally, Section V presents our conclusions.

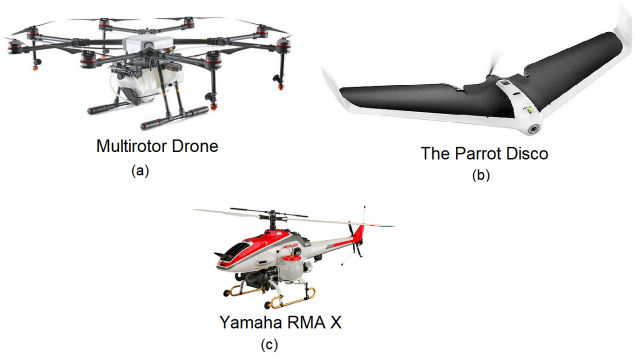

Figure 1. UAV types: multi-rotor (a), fixed wing (b), and single-rotor (c).

II. Background

A. Generic Model of UAV Systems

Reg Austin [14] defines UAVs as a system comprising a number of subsystems, including the aircraft (often referred to as a UAV or unmanned air vehicle), its payloads, the Ground Control Station (GCS) (and, often, other remote stations), aircraft launch and recovery subsystems, where applicable, support subsystems, communication subsystems, and transport subsystems. UAVs have different shapes and models to meet the various tasks assigned to them, such as fixed-wing, single rotor, and multi-rotor, as illustrated in Fig. 1. However, their functional structure has a fixed standard, as shown in Fig. 2. Therefore, finding a security vulnerability in one model might lead to exploiting the same vulnerability in a wide range of different systems [15], [16].

B. Cyber-Threats

A cyber-threat in UAVs represents a malicious action by an attacker with the goal of damaging the surrounding environment or causing financial losses, where the UAV is typically deployed [17]. In particular, with some of these UAVs available to the general public, ensuring their secure operation still remains an open research question, especially when considering the sensitivity of previous cyber-attacks described in the literature [18], [19].

One notable example is the control of deadly weapons as with the US military RQ-170 Sentinel stealth aircraft; it was intercepted and brought down by the Iranian forces in late 2011 during one of the US military operations over

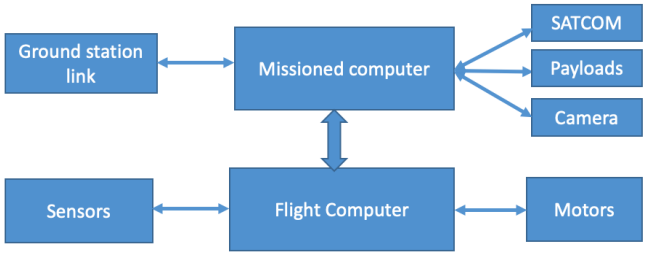

Figure 2. Functional structure of UAVs.

the Iranian territory [18]. In 2018, Israel released footage for one of its helicopters shooting down an Iranian replica model of the US hijacked drone [19]. Further interest in UAV cyber-security has been raised following this attack. For example, Nils Rodday [20], a cyber-security analyst, was able to hack UAVs utilized by the police using a man-in-the-middle attack by injecting control commands to interact with the UAV. As a result of previous attacks, UAVs can be a dangerous weapon in the wrong hands. Obviously, cyber-attack threats exceeded the cyber-space barrier as observed by Tarabay, Lee and Frew [18], [19]. Therefore, enhancing the security and resilience of UAV software has become a vital homeland security mission, mainly due to the possible damage of cyber-attacks from the deployed UAVs.

C. Verification of Security in UAVs

The UAV components are typically connected together to enable secure and fast communication; if one component fails, the entire system can be susceptible to malicious attacks [14]. In this respect, various approaches have been taken to automatically verify the correctness of UAVs software. In particular, following the RQ-170 UAV accident in 2011, where Iran claimed hacking the sophisticated U.S. UAV [21], a group of researchers from the University of Texas proposed an usual exercise to the U.S. Department of Homeland Security; specifically simulated GPS signals were transmitted over the air from 620 m, where the spoofer induced the capture GPS receiver to produce position and velocity solutions, which falsely indicated the UAV position [22]. A similar study conducted in early 2018 performed a successful side-channel attack to leverage physical stimuli [23]. The authors were able to detect in real-time whether the UAV's camera is directed towards a target or not, by analyzing the encrypted communication channel, which was transmitted from a real UAV. These prior studies were able to highlight GPS weaknesses, but they did not cover the UAV security issues w.r.t. all involved software elements, mainly when zero-day vulnerabilities are associated with the respective UAV outputs, i.e., a real UAV software bug that is unknown to the vendor responsible for patching or otherwise fixing the bug.

Other related studies focus on automated testing [24] and model-checking the behavior of UAV systems [8], [25]. For example, a verification tool named as Digital System Verifier (DSVerifier) [8] formally checks digital-system implementation issues, in order to investigate problems that emerge in digital control software designed for UAV attitude systems (i.e., software errors caused by finite word-length effects). Similar work also focuses on low-level implementation aspects, where Sirigineedi et al. [25] applied a formal modeling language called SMV to multiple-UAV missions by means of Kripke structures and formal verification of some of the mission properties typically expressed in Computational Tree Logic. In this particular study, a deadlock has been found and the trace generated by SMV has been successfully simulated. Note that these prior studies concentrate mainly on the low-level implementation aspects of how UAVs execute pilot commands. By contrast, we focus our approach on the high-level application of UAVs software, which is typically hosted by the firmware embedded in UAVs.

Despite the previously discussed limitations, BMC and Fuzzing techniques have been successfully used to verify the correctness of digital circuits, security, and communication protocols [25], [26]. However, given the current knowledge in ensuring security of UAVs, the combination of fuzzing and BMC techniques have not been used before for detecting security vulnerabilities in UAV software (e.g., buffer overflow, dereferencing of null pointers, and pointers pointing to unallocated memory regions). UAV software is used for mapping, aerial analysis and to get optimized images. In this study, we propose to use both techniques to detect security vulnerabilities in real-world UAV software.

III. Finding Software Vulnerabilities in UAVs Using Software Verification

A. Software In-The-Loop

UAV software has a crucial role to operate, manage, and provide a programmatic access to the connected UAV system. In particular, before a given UAV starts its mission, the missioned computer, as illustrated in Fig. 2, exports data required for this mission from a computer running the flight planning software. Then, the flight planning software allows the operator to set the required flight zone (waypoint mission engine), where the UAV will follow this route throughout its mission instead of using a traditional remote controller directly [24].

Dronekit[1] is an open-source software project, which allows one to command a UAV using Python programming language [27]; it enables the pilot to manage and direct control over the UAV movement and operation, as illustrated in Fig. 3, where one can connect to the UAV via a User Datagram Protocol (UDP) endpoint (line 3) with the goal of

[1]https://github.com/dronekit/dronekit-python

```
from dronekit import connect
# Connect to UDP endpoint.
vehicle = connect('127.0.0.1:14550', wait_ready=True)
# Use returned Vehicle object to query device state:
print("Mode: %s" % vehicle.mode.name)
```

Figure 3. Python script to connect to a vehicle (real or simulated).

gaining control of the UAV by means of the "vehicle" object. In particular, UDP allows establishing a low-latency and loss-tolerating connection between the pilot and the UAV. This control process relies on the planning software inside the UAVs system, which in some cases the software might be permanently connected to the pilot controlling system (e.g., Remote Controller or GCS) due to live feedback or for real-time streaming.

Our main research goal is to investigate in depth open-source UAVs code (e.g., DJI Tello[2] and Parrot Bebop[3]) to search for potential security vulnerabilities. For example, Fig. 4 shows a simple Python code to read and view various data status of Tello UAV. In particular, this Python code imports and defines the required libraries (lines from 1 to 3) and then connects the GCS to the UAV by using the predefined port and IP address in lines 11 and 12. As we can see from lines 15 to 25, the UAV will acknowledge the pilot commands and print the Tello current status. If an attacker is able to scan and locate the IP address that this particular UAV has used, then he/she would be able to easily intercept the data transmitted, inject a malicious code or take the drone out of service using a denial of service attack, which can lead the UAV to a crash, thus making it inaccessible. In order to detect potential security vulnerabilities in UAV software, we provide here an initial insight of how to combine BMC and fuzzing techniques with the goal of exploring the system state-space to ensure safe and secure operations of UAVs.

1) Illustrative Example Using UAV swarm: Throughout this paper, we use an illustrative example from UAV swarm, which consists of multiple UAVs to autonomously make decisions based on shared information; the safe and secure operation of multiple UAVs is particularly relevant since they have the potential to revolutionize the dynamics of conflict. In early 2019, we participated in a competitive-exercise[4], with five different UK universities; the main goal of this event consisted of teams from across the UK to compete against each other in a game of offense (red team) and defense (blue team) using swarms of UAVs, as illustrated in Fig. 5. As a result, this competition allowed us to highlight aspects of how to protect urban spaces from UAV swarms, which is a serious concern of modern society. This competition was sponsored by the British multinational defense, security, and aerospace company (BAE). Solutions

[2]https://github.com/dji-sdk/Tello-Python
[3]https://github.com/amymcgovern/pyparrot
[4]https://www.youtube.com/watch?v=dyyaY1VXqL4

```
1  import socket  \label{python:import11}
2  from time import sleep
3  import curses
4  INTERVAL = 0.2
5  ...
6  local_ip = ''
7  local_port = 8890
8  socket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
9  # socket for sending cmd
10 socket.bind((local_ip, local_port))
11 tello_ip = '192.168.10.1'
12 tello_port = 8889
13 tello_adderss = (tello_ip, tello_port)
14 socket.sendto('command'.encode('utf-8'), tello_adderss)
15 try:
16   index = 0
17   while True:
18     index += 1
19     response, ip = socket.recvfrom(1024)
20     if response == 'ok':
21       continue
22     out = response.replace(';', ';\n')
23     out = 'Tello State:\n' + out
24     report(out)
25     sleep(INTERVAL)
26 ...
```

Figure 4. Python code fragment to read and view various data status of Tello UAV.

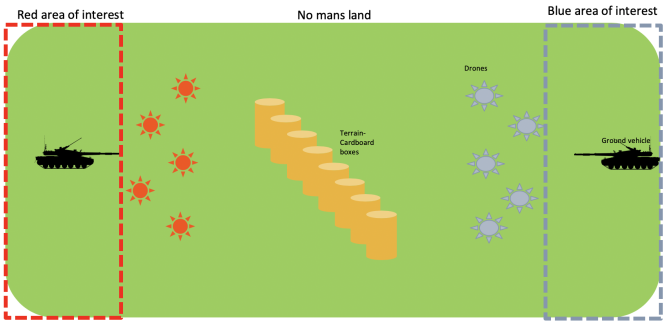

Figure 5. UAV Swarm Competition.

developed by industry, such as the "jamming guns" and single "UAV catchers", fall short of what would be required to defend against a large automated UAV swarm attack. For this particular illustrative example, using software verification and the UAV connection weakness, we were able to perform a successful cyber-attack against UAV models by scanning the radio frequencies and targeting the unwanted UAVs with just a raspberry-pi, a Linux OS installed on and 2.4 *GHz* antennas, as reported in our experimental evaluation.

B. Verifying UAV Software using Fuzzing and BMC

We describe our novel verification approach approach called "UAV Fuzzer" to check for security vulnerabilities in UAVs. In particular, we check for user-specified assertions, buffer overflow, memory safety, division by zero, and arithmetic under- and overflow. Our verification approach consists of running a fuzzer engine using pre-collected test

cases $TC = \{tc_1, tc_2, \ldots, tc_n\}$, where n represents the total number of pre-collected test cases, with the goal of initially exploring the state-space of the UAV software operation. Note that a test case tc_i used by our approach is similar to a real valid data, but it must contain a problem on it, or also called "anomalies". For example, to fuzz UAV software, a test case should be a connection between the UAV and GCS, so the mutated version generated of such a similar connection is called a test case tc.

In our "UAV Fuzzer", we keep track of each computation path of the program $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$, which represents the program unfolding for a bound k and a security property ϕ initially explored by our fuzzer. If our fuzzer engine gets stuck due to the mutations generated are not suited enough to the new state transition, our BMC tool runs against the target software to symbolically explore its uncovered state-space with the goal of checking the unexplored execution paths in Π of the UAV software. The idea behind BMC is to check the negation of a given property ϕ at a given depth k, i.e., given a transition system *M*, a property ϕ, and a limit of iterations *k*, BMC unfolds a given system *k* times and converts it into a Verification Condition (VC) ψ, such that ψ is *satisfiable* if and only if ϕ has a counterexample (cex) of depth less than or equal to *k*.

We formally describe our verification algorithms "UAV Fuzzer" by assuming that a given program P under verification is a state transition system M. In M, a state $s \in S$ consists of the value of the program counter and the values of all program variables. A predicate $init_P(s)$ denotes that s is an initial state, $tr_p(s_i, s_j) \in T$ is a transition relation from s_i to s_j, $\phi(s)$ is the formula encoding for states satisfying a safety property, and $\psi(s)$ is the formula encoding for states satisfying a completeness threshold [28], which is equal to the maximum number of loop iterations occurring in P. For convenience, we define an error state ϵ, reachable if there exists a property violation in the program P. A counterexample cex^k is a sequence of states of length k from an initial state s_1 to ϵ. The main steps for our proposed verification algorithm are described in Algorithm 1.

As an illustrative example, consider the code fragment shown in Fig. 4. First, our UAV fuzzer starts by defining new data based on the input expected by our targeted model (Tello UAV). As an example, Fig. 6 shows a valid test case generated by our UAV fuzzer, which is based on the module specification for the Tello UAV; this test case expects the IP address and specific port before launching the drone to start flying. Second, our UAV fuzzer engine starts exploring and running the UAV software (cf. Fig. 4) with the generated test-cases (cf. Fig. 6) and then it records each execution path π_i that has been explored. Third, when the UAV fuzzer engine reaches a complex condition and struggles to find its next path (line 5 of Fig. 6), UAV fuzzer will attempt to reconstruct the following path using BMC, which stores the current fuzzing transactions and restores

Algorithm 1 UAV Fuzzer

1: Define pre-collected test cases $TC = \{tc_1, tc_2, \ldots, tc_n\}$ to be employed by the fuzzing engine.
2: Fuzzer engine begins to explore each execution path π_i starting from an initial state s and produces malformed inputs $I = \{\iota_1, \iota_2, \ldots, \iota_n\}$ to test for potential security vulnerabilities.
3: Store each π_i that has been verified and repeat step 2 until the fuzzer engine either reaches a crashing point or it cannot explore the next π_i in Π due to complex guard checks.
4: Run BMC to verify the remaining execution paths in Π that have not been previously explored by our fuzzer engine in steps 2 and 3.
5: Repeat step 4 until BMC falsifies or verifies ϕ or it exhausts time and memory limits.
6: Once BMC completely verifies the UAV code in step 5, it returns "false" if a property violation is found together with cex^k, "true" if it is able to prove correctness, or "unknown".

the next path symbolically. Lastly, BMC will check for any further exception that occurs as a result of its execution. Additionally, UAV fuzzer can report the code coverage achieved during the testing process, and thus provide a better understanding of code coverage status.

```
1   while True:
2     index %= 1
3 # + replaced with %
4     response, ip = socket.recvfrom(1024)
5     if response == 'ok'
6       continue
```

Figure 6. Test case from the Tello UAV embedded software.

C. UAV Communication Channel

An UAV has a radio to enable and facilitate remote communication between the GCS and the UAV. In addition, it consists of different electronic components, which interact autonomously with a goldmine of data transmitted over the air during its flight's missions; this makes the communication channel in UAVs an ideal target for a remote cyber-threat. Therefore, ensuring secure (bug-free) software, together with a secure communication channel, emerges as a priority in successful deployment of any UAV system.

A successful false-data injection attack was demonstrated by Strohmeier et al. [29], which had devastating effects on the UAV system. The authors were able to successfully inject valid-looking messages, which are well-formed with reasonable data into the Automatic Dependent Surveillance-Broadcast (ADS-B) protocol. Note that this protocol is currently the only means of air traffic surveillance today,

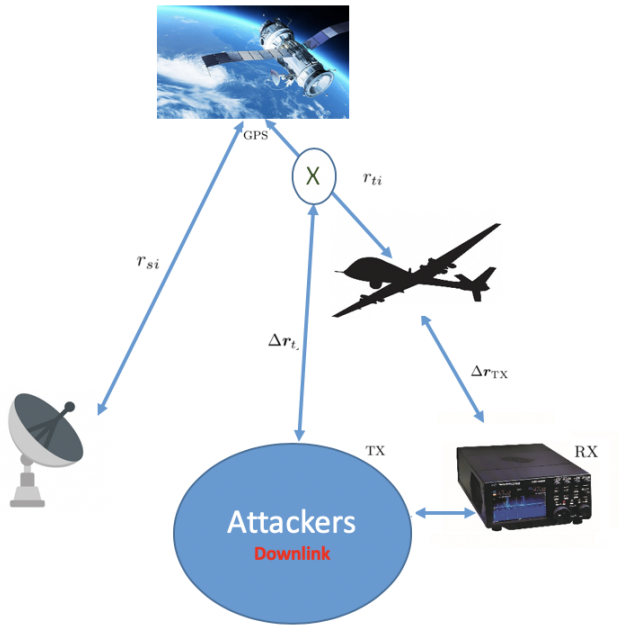

Figure 7. GPS-satellite-signal is overlaid by a spoofed GPS-signal.

where Europe and US must comply with the mandatory use of this insecure protocol by the end of 2020 [29].

To investigate this layer further, we used Software-Defined Radio (SDR) system to receive, transmit, and analyze the UAV operational connection system (e.g., Ku-Band and WiFi). We have also investigated the information exchanged between UAV sensors and the surrounding environment for any potential security vulnerabilities (e.g., GPS Spoofing), as illustrated in Fig. 7. The signal that comes from the satellite is weak; hence, if an attacker uses a local transmitter under the same frequency, this signal would be stronger than the original satellite signal. As a result, the spoofed GPS-signal will override current satellite-signal, thereby leading to spoof a fake position for the UAV targeted. In this particular case, the UAV would then be hijacked and put in hold, waiting for the attacker's next command. Therefore, verifying the UAV software to build practical software systems with strong safety and security guarantees is highly needed in practice.

Our GPS spoofing attack is described in Algorithm 2. Here spoofer refers to a Full-Duplex device that is used to attack a particular UAV system to crash or take control of the UAVs. In our experimental evaluation with the UAV models DJI Tello and Parrot Bebop 2, we were able to perform a successful attack under 2.4 *GHz* to the target system. First, we were able to detect the drone frequency by one of the antennae, which configured to monitor the active 2.4 GHz connection referred to as RX. The targeted drones are allocated based on identifying the drone default mac addresses owned by the Parrot company[5] and their unique SSID. Second, the other antenna was used to transmit the attacker new data referred to as TX. The distance between the attacker equipment and the target (r_{si} and r_{ti}) is vital for this attack, since the antenna/system strength and the delay caused are all taken into consideration during the attack.

Algorithm 2 GPS Spoofing Attack

1: The spoofer device should be located with the nominated antennas (i.e., 2.4 *GHz*).
2: The antenna configured on monitoring mode (*RX*) to detect the authentic signal from the available GPS satellites.
3: The vectors $\Delta \mathbf{r}_{TX}$ and $\Delta \mathbf{r}_t$ are for the antenna (*TX*) coordinates which distributed in a 3-dimensional array.
4: r_{si} and r_{ti} are the respective distances from the GPS satellite.

IV. Preliminary Experimental Evaluation

We have performed a preliminary evaluation of our proposed verification approach to detect security vulnerabilities in UAVs. Our proposed method was implemented in a tool called DepthK [30], using BMC technique and invariant generators such as PIPS [31] and PAGAI [32]. PIPS is an inter-procedural source-to-source compiler framework for C and Fortran programs, PAGAI is a tool for automatic static analysis that is able to generate inductive invariants, both rely on a polyhedral abstraction of program behavior for infering invariants. We have evaluated the DepthK tool over a set of standard C benchmarks, which share common features of UAV code (e.g., concurrency and arithmetic operations). We have also evaluated our fuzzer engine to test a PDF software with the goal of checking its efficiency and efficacy to identify bugs. Lastly, we present our results in the swarm competition promoted by BAE systems.

A. Description of Benchmarks

The International Software Verification Competition (SV-COMP) [33], where DepthK participated, was run on a Linux Ubuntu 18.04 OS, 15 GB of RAM memory, a run-time limit of 15 minutes for each verification task and eight processing units of $i7-4790$ CPU. The SV-COMP's benchmarks used in this experimental evaluation include: *ReachSafety*, which contains benchmarks for checking reachability of an error location; *MemSafety*, which presents benchmarks for checking memory safety; *ConcurrencySafety*, which provides benchmarks for checking concurrency problems; *Overflows*, which is composed of benchmarks for checking whether variables of signed-integers type overflow; *Termination*, which contains benchmarks for which termination

[5]http://standards-oui.ieee.org/oui/oui.txt

should be decided; *SoftwareSystems*, which provides benchmarks from real software systems.

Our fuzzing experiments were ran on MacBook Pro laptop with 2.9 GHz Intel Core i7 processor and 16 GB of memory. We ran our fuzzing engine for at most of 12 hours for each single binary file. We analyzed and replayed the testing result after a crash was reported or after the fuzzer hit the time limit. To analyze the radio frequencies, we configured/compiled the required software for this purpose (e.g. bladerf, GQRX, OsmoSDR, and GNU Radio tool) using bladerf x40 device, ALFA high gain USB Wireless adapter and 2.4 *GHz* antennas. Additionally, we used the open-source UAVs code DJI Tello and Parrot Bebop.

B. Objectives

The impact of our study is a novel insight on the UAV security potential risks. In summary, our evaluation has the following three experimental questions to answer:

EQ1 **(Localization)** Can DepthK help us understand the security vulnerabilities that have been detected?

EQ2 **(Detection)** Can generational or mutational fuzzers be further developed to detect vulnerabilities in real-world software?

EQ3 **(Cyber-attacks)** Are we able to perform successful cyber-attacks in commercial UAVs?

C. Results

1) SV-COMP: Concurrency bugs in UAVs are one of the most difficult vulnerabilities to verify [25]. Our software verifier DepthK [30] has been used to verify and falsify safety properties in C programs, using BMC and *k*-induction proof rule techniques. In late 2018, we participated with the DepthK tool in SV-COMP 2019[6] against other software verifiers. Our verifier showed promising results over thousands of verification tasks, which are of particular interest to UAVs security (*e.g., Concurrency Safety* and *Overflows* categories), which answers **EQ1**.

Concurrency Safety category, which consists of 1082 benchmarks of concurrency problems, is one of the many categories verifiers run over; DepthK was able to accurately detect 966 problems from this category. For the *Overflows* category, which consists of 359 benchmarks for different signed-integers overflow bugs, DepthK was able to detect 167 problems. These results are summarized in Table I. A task counts as *correct true* if it does not contain any reachable error location or assertion violation, and the tool reports "safe"; however, if the tool reports "unsafe", it counts as *incorrect true*. Similarly, a task counts as *correct false* if it does contain a reachable violation, and the tool reports "unsafe", together with a confirmed witness (path to failure); otherwise, it counts as *incorrect false* accordingly. Dirk Beyer [33] shows DepthK's results when compared with other verifiers in SV-COMP 2019.

[6]https://sv-comp.sosy-lab.org/2019/

Table I
DEPTHK RESULTS IN SV-COMP 2019.

Category list	Correct True	Correct False	Incorrect Results	Unknown
Concurrency Safety	194	772	20	96
Overflows	17	150	0	192

Table II
FUZZING APPROACHES COMPARISON.

Fuzzing Approaches	Target	Time	Faults
Generational Fuzzer	Sumatra PDF	45 hours	70
Mutational Fuzzer	Sumatra PDF	15 hours	23

2) Fuzzing Approach: According to a prior study [9], the generalizing fuzzing approach leads to a better result in discovering and recording software vulnerabilities compared with the mutational fuzzing approach if the test cases used in the fuzzing experiment are taken into account, which answers **EQ2**. Our experimental results applied to a PDF software called Sumatra PDF[7], which was chosen for evaluation purposes, are shown in Table II. Here, the generational fuzzer was able to detect 70 faults in 45 hours in the Sumatra PDF, while the mutational fuzzer was able to detect 23 in 15 hours.

3) UAV Swarm Competition: As part of our participation at the UAV swarm competition sponsored by (BAE)[8], penetration testing was performed against the UAV both connection and software system, in which we were able to perform successful cyber-attacks, which answers **EQ3**. These attacks led to deliberately crash UAVs or to take control of different non-encrypted UAV systems (e.g., Tello and Parrot Bebop 2). This was achieved by sending connection requests to shut down a UAV CPU, thereby sending packets of data that exceed the capacity allocated by the buffer of the UAV's flight application and by sending a fake information packet to the device's controller. These results are summarized in Table III, where we describe the employed UAV models and tools and whether we were able to obtain full control or crash. Note that due to the limitations of the competition, DepthK tool was not employed during the BAE competition; however, exploiting potential UAV software vulnerabilities is still a continuous research, where we intend to further exploit DepthK.

D. Threats to Validity

Benchmark selection: We report the evaluation of our approach over a set of real-world benchmarks, where the UAVs share the same component structure. Nevertheless, this set is limited within our research scope and the experiment results may not generalize to other models because other UAV models have a proprietary source-code. Additionally,

[7]https://www.sumatrapdfreader.org/free-pdf-reader.html

[8]https://www.cranfield.ac.uk/press/news-2019/bae-competition-challenges-students-to-counter-threat--from-uavs

Table III
RESULTS OF THE UAV SWARM COMPETITION.

Vulnerability Type	UAV Model	Tool	Result
Spoofing	DJI Tello	Wi-Fi transmitter	Full Control
Denial of service			Full Control
Spoofing	Parrot bebop 2	Wi-Fi transmitter	Full Control
Denial of service			Crash

we have not evaluated our verification approach using real UAV code written in Python, which is our main goal for future research.

Radio Spectrum: The frequencies we report on our evaluation were between 2.4 *GHz* and 5.8 *GHz*, as the two most common ranges for civilian UAVs; however, the radio regulations in the UK are complicated (e.g., we are required to be either licensed or exempted from licensing for any transmission over the air).

V. CONCLUSIONS AND FUTURE WORK

Our ultimate goal is to develop a UAV fuzzer to be introduced as mainstream into the current UAV programming system, in order to build practical software systems robust to cyber-attacks. We have reported here an initial insight of our verification approach and some preliminary results using similar software typically used by UAVs. In order to achieve our ultimate goal, we have various tasks planned as follows:

- **Vulnerability Assessment**: Identify and implement simple cyber-attacks from a single point of attack against different UAV models. We will continue investigating Python vulnerabilities at the high-level system (e.g., UAV applications) and whether UAVs software is exploitable to those security vulnerabilities.
- **Python Fuzzer**: We will develop an automated python fuzzer by analyzing how to convert the UAV command packets into a fuzzing ones, in order to produce test cases, which are amenable to our proposed fuzzer.
- **GPS Analysis**: We identified based on numerical analysis on GPS, the cyber-attack UAVs might be vulnerable from. This investigation will continue to develop and simulate a GPS attack applied to a real UAV system.
- **Implementation**: Apply our proposed verification approach to test real-world software vulnerabilities, which can be implemented during the software development life-cycle to design a cyber-secure architecture.
- **Evaluation and Application**: Evaluate our proposed approach using real-world UAV implementation software. We will also compare our approach in different stages to check its effectiveness and efficiency.

ACKNOWLEDGMENT

Mustafa A. Mustafa is funded by the Dame Kathleen Ollerenshaw Fellowship awarded by The University of Manchester.

REFERENCES

[1] PwC, "Drones could add £42bn to UK GDP by 2030 - PwC research," 2018. [Online]. Available: https://www.pwc.co.uk/press-room/press-releases/pwc-uk-drones-report.html

[2] PwC., "Gatwick airport drones disruption wasn't all for nothing, UK police insist," 2018. [Online]. Available: https://edition.cnn.com/2018/12/24/uk/gatwick-airport-drones-investigation-gbr-intl/index.html

[3] A. House, *Drone Safety Risk: An assessment*. Civil Aviation Authority, 2018.

[4] M. Farooq, M. Waseem, A. Khairi, and S. Mazhar, "Article: A critical analysis on the security concerns of internet of things (iot)," *IJCA*, vol. 111, no. 7, pp. 1–6, February 2015.

[5] A. Biere, *Handbook Of Satisfiability*. IOS Press, 2009, vol. 185, ch. 26.

[6] M. R. Gadelha, F. R. Monteiro, J. Morse, L. C. Cordeiro, B. Fischer, and D. A. Nicole, "ESBMC 5.0: an industrial-strength C model checker," in *ASE*, 2018, pp. 888–891. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3238147.3240481

[7] B. P. Miller, G. Cooksey, and F. Moore, "An empirical study of the robustness of macos applications using random testing," in *RT*. ACM, 2006, pp. 46–54.

[8] L. Chaves, I. Bessa, H. Ismail, A. Frutuoso, L. Cordeiro, and E. de Lima Filho, "DSVerifier-Aided Verification Applied to Attitude Control Software in Unmanned Aerial Vehicles," *IEEE Transactions on Reliability*, vol. 67, 2018.

[9] O. Alhawi, A. Akinbi, and A. Dehghantanha, "Evaluation and Application of Two Fuzzing Approaches for Security Testing of IoT Applications," in *Handbook of Big Data and IoT Security*, 2019, pp. 301–327. [Online]. Available: http://link.springer.com/10.1007/978-3-030-10543-3{_}13

[10] B. Cook, K. Khazem, D. Kroening, S. Tasiran, M. Tautschnig, and M. R. Tuttle, "Model checking boot code from AWS data centers," in *Computer Aided Verification*, vol. 10982. Springer, 2018, pp. 467–486.

[11] P. Godefroid, M. Y. Levin, and D. Molnar, "Automated Whitebox Fuzz Testing," *Queue - Networks*, 2012. [Online]. Available: https://www.eecs.northwestern.edu/{~}robby/courses/395-495-2017-winter/ndss2008.pdf

[12] S. Artzi, A. Kie zun, J. Dolby, F. Tip, D. Dig, A. Paradkar, and M. D. Ernst, "Finding Bugs in Dynamic Web Applications," *ISSTA*, pp. Pages 261–272, 2008. [Online]. Available: http://www.htmlkit.com

[13] Corina S. Pasareanu, "Using Symbolic (Java) PathFinder at NASA," *Nasa*, 2010. [Online]. Available: https://pdfs.semanticscholar.org/8933/ac2dbeb3ccd8cf3e393d8dbb22ccc4452b64.pdf

[14] R. Austin, "UNMANNED AIRCRAFT SYSTEMS UAVS DESIGN, DEVELOPMENT AND DEPLOYMENT," *A John Wiley and Sons*, vol. 54, 2011. [Online]. Available: https://archive.org/details/UnmannedAircraftSystemsUAS

[15] V. Dey, V. Pudi, A. Chattopadhyay, and Y. Elovici, "Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study," in *VLSID*. IEEE Computer Society, 2018, pp. 398–403.

[16] S. Frei, M. May, U. Fiedler, and B. Plattner, "Large-scale vulnerability analysis," in *LSAD '06*, 2006, pp. 131–138.

[17] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam, "Cyber security threat analysis and modeling of an unmanned aerial vehicle system," in *HST*, Nov 2012, pp. 585–590.

[18] Joanna Frew, "An overview of new armed drone operators The Next Generation," Drone Wars UK, Oxford, Tech. Rep., 2018. [Online]. Available: www.dronewars.net

[19] O. L. Jamie Tarabay and I. Lee, "Israel: Iranian drone we shot down was based on captured US drone - CNN," 2018. [Online]. Available: https://edition.cnn.com/2018/02/12/middleeast/israel-iran-drone-intl/

[20] N. M. Rodday, R. d. O. Schmidt, and A. Pras, "Exploring security vulnerabilities of unmanned aerial vehicles," in *NOMS*, April 2016, pp. 993–994.

[21] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys, "Unmanned aircraft capture and control via gps spoofing," *Journal of Field Robotics*, vol. 31, no. 4, pp. 617–636, 2014. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21513

[22] M. L. Psiaki, B. W. O'Hanlon, J. A. Bhatti, D. P. Shepard, and T. E. Humphreys, "Gps spoofing detection via dual-receiver correlation of military signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 49, no. 4, pp. 2250–2267, OCTOBER 2013.

[23] B. Nassi, R. Ben-Netanel, A. Shamir, and Y. Elovici, "Drones' cryptanalysis - smashing cryptography with a flicker," in *2019 2019 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019, pp. 832–849. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP.2019.00051

[24] M. A. Day, M. R. Clement, J. D. Russo, D. Davis, and T. H. Chung, "Multi-uav software systems and simulation architecture," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2015, pp. 426–435.

[25] G. Sirigineedi, A. Tsourdos, R. Żbikowski, and B. A. White, "Modelling and Verification of Multiple UAV Mission Using SMV," *EPTCS*, vol. 20, pp. 22–33, 2010. [Online]. Available: https://dspace.lib.cranfield.ac.uk/handle/1826/3978

[26] K. Domin, I. Symeonidis, and E. Marin, "Security analysis of the drone communication protocol: Fuzzing the mavlink protocol," 2016.

[27] G. Van Rossum and F. L. Drake Jr, *Python tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands, 1995.

[28] D. Kroening, J. Ouaknine, O. Strichman, T. Wahl, and J. Worrell, "Linear Completeness Thresholds For Bounded Model Checking," in *Computer-Aided Verification*, ser. LNCS, vol. 6806, 2011, pp. 557–572.

[29] M. Strohmeier, V. Lenders, and I. Martinovic, "Intrusion detection for airborne communication using phy-layer information," in *DIMVA*, 2015, pp. 67–77.

[30] W. Rocha, H. Rocha, H. Ismail, L. Cordeiro, and B. Fischer, "Depthk: A k-induction verifier based on invariant inference for c programs," in *TACAS, LNCS 10206*, 2017, pp. 360–364.

[31] M. ParisTech, "PIPS: Automatic Parallelizer and Code Transformation Framework," Available at http://pips4u.org, 2013.

[32] J. Henry, D. Monniaux, and M. Moy, "PAGAI: A Path Sensitive Static Analyser," in *Electron. Notes Theor. Comput. Sci.* Elsevier Science Publishers B. V., 2012, pp. 15–25.

[33] D. Beyer, "Automatic verification of C and java programs: SV-COMP 2019," in *TACAS, LNCS 11429*, 2019, pp. 133–155.