Finding Security Vulnerabilities in Unmanned Aerial Vehicles Using Software Verification
(Technical Report)

Omar M. Alhawi[1], Mustafa A. Mustafa[1,2], and Lucas C. Cordeiro[1]

[1] University of Manchester, United Kingdom
[2] imec-COSIC KU Leuven, Belgium
{omar.alhawi, mustafa.mustafa, lucas.cordeiro}@ manchester.ac.uk

Abstract. The proliferation of Unmanned Aerial Vehicles (UAVs) embedded with vulnerable monolithic software, involving concurrency and fragile communication links, has recently raised serious concerns about their security. Recent studies show that a 2kg UAV can cause a critical damage to a passenger jet windscreen. However, verifying security in UAV software based on traditional testing remains an open challenge mainly due to scalability and deployment issue. Here we investigate the application of software verification techniques; in particular, existing software analyzers and verifiers, which implement fuzzing and bounded model checking techniques, to detect security vulnerabilities in typical UAVs. We also investigate fragility aspects related to the UAV communication link since all remaining UAV components (e.g., position, velocity and attitude control) heavily depend on it. Our preliminary results show real cyber-threats with the possibility of exploiting further security vulnerabilities in real-world UAV software in the foreseeable future.

Keywords: Software Verification · Software Testing · UAV · Security.

1 Introduction

Unmanned Aerial Vehicles (UAVs), also sometimes referred 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 important devices in various aspects of life and work in the UK, thereby leading to GBP 42bn increase in the UK's gross domestic product and 628,000 jobs in its economy [24].

With this ever growing interest also comes a growing danger of cyber-attacks, which can pose high safety risks to large airplanes and ground installations, as recently witnessed at the Gatwick airport in late 2018, when unknown UAVs flying close to the runways caused disruption and cancellation of hundreds of flights due to safety concerns.[3] Another popular example of cyber attack is one

[3] https://edition.cnn.com/2018/12/24/uk/gatwick-airport-drones-investigation-gbr-intl/index.html

related to when the Lebanese militant group (Hezbollah),[4] produced several footages of aerial view in late 2010, thus claiming to be intercepted from one of the Israeli surveillance UAVs over Lebanon and accusing Israel of being behind the Lebanese prime minster assassination. Therefore, it remains an open question whether the CIA[5] triad principles will be maintained during UAVs software development life-cycle. UAVs typically demand high-quality software, in order to meet its target system's requirements. In particular, 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, where they are usually needed to guarantee both user requirements and system behavior.

Although Bounded Model Checking (BMC) was introduced nearly two decades ago, it 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 finding real bugs, they typically suffer from the state-space explosion problem [13].

Fuzzing is a successful testing techniques that can create a substantial amount of random data to discover security vulnerabilities in real-world software [20], but subtle bugs in UAVs might still go unnoticed due to the large state-space exploration, as recently reported by Chaves et al. [6]. Additionally, according to Alhawi et al. [1], 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, including automatic detection of bugs and security vulnerabilities, recovery of corrupt documents, patch generation, and automatic debugging, which have been industrially adopted by large companies, including but not limited to Amazon Web Service (CBMC [7]), Microsoft (SAGE [14]), IBM (Apollo [2]), and NASA (Symbolic PathFinder [8]). For example, the SAGE fuzzer has already discovered more than 30 new bugs in large shipped Windows applications [14]. Nonetheless, an open research question consists in whether these techniques can be effective in terms of correctness and performance to verify UAVs applications.

In order to make an impact to ensure UAV security, our research will investigate both fuzzing and BMC techniques to automatically detect security vulnerabilities in real-world UAV software. Thus, our main goal is to build a software system that has immunity from cyber-attacks and thus ultimately improve software reliability. According to the current cyber-attacks profile w.r.t.

[4] `https://www.military.com/defensetech/2010/08/10/hezbollah-claims-it-hacked-israeli-drone-video-feeds`

[5] *Confidentiality*, *Integrity* and *Availability*, also known as the CIA triad, is a model designed to guide policies for information security [11].

advanced UAVs, it becomes clear that the current civilian UAVs in the market are not secure enough even from simple cyber-attacks. In order 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, which led to gain a full unauthorized control or cause UAV to crash. We also show that pre-knowledge of the receptiveness of the UAV system components to manipulation is all what attackers need to know during their reconnaissance phase before exploiting UAV weaknesses.

Contributions. Our main contribution is to propose a new approach for detecting and exploiting security vulnerabilities in UAVs. In particular, 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 major contributions:

- Provide an initial insight into fuzzing and BMC when applied to UAV software;
- Identify different security vulnerabilities that UAVs are susceptible from. Here we perform real cyber-attacks against different UAV models with the goal of highlighting the cyber-threats related to them;
- Propose a preliminary verification approach called "UAV fuzzer" to be compatible with the type of UAV software currently being developed in industry with the goal of detecting their vulnerabilities.

2 Background

2.1 Generic Model of UAV Systems

Reg Austin [3] defines UAVs as a system comprising a number of sub-systems, 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 sub-systems, 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 illustrated 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 [10, 12].

2.2 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 employed [17]. In particular, with some of these UAVs available to the general public, ensuring their secure operation remains an open research question, especially when considering the sensitivity of previous cyber-attacks in literature [16, 18].

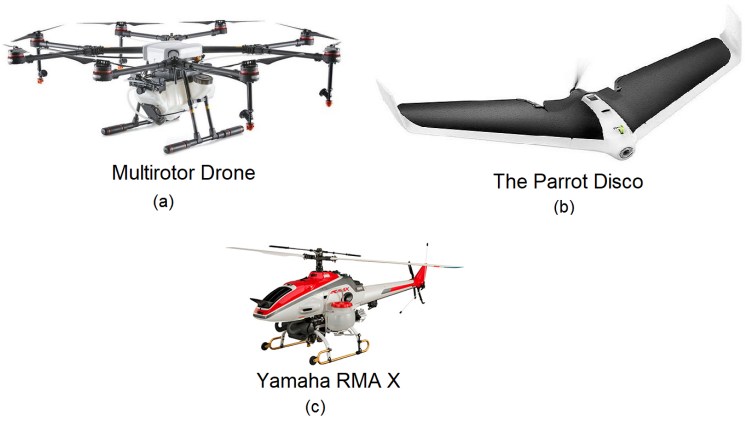

Fig. 1. UAV types: multi-rotor (a), fixed wing (b), and single-rotor (c).

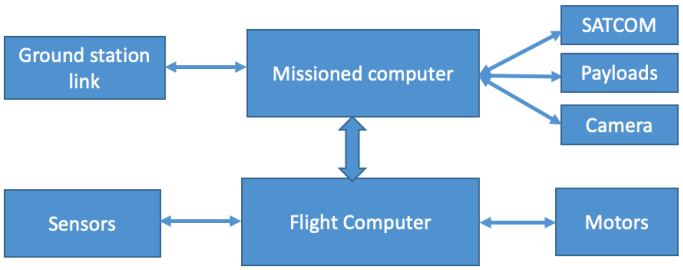

Fig. 2. Functional structure of UAVs.

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 late 2011 during one of the US military operations over 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 [16]. Further interest in UAV cyber-security has been raised following this attack. For example, Nils Rodday [22], 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 [16, 18]. 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.

2.3 Verification of Security in UAVs

The UAV components are typically networked together to enable secure and fast communication. Therefore, if one component fails, the entire system can be susceptible to malicious attacks. 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 [19], a group of researchers from the University of Texas proposed an usual exercise to the U.S. Department of Homeland Security; 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 [4]. A similar study conducted in early 2018 performed a successful side-channel attack to leverage physical stimuli [21]. 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. As a consequence, these prior studies were able to highlight GPS weaknesses, but they did not cover the UAV security issues regarding all involved software elements, mainly when those zero-day vulnerabilities are associated with the respective UAV outputs.

Other related studies focus on automated testing [9] and model-checking the behavior of UAV systems [6, 26]. For example, a recent verification tool named as Digital System Verifier (DSVerifier) [6] 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. [26] 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. However, note that these prior studies concentrate mainly on the low-level implementation aspects of how UAVs execute pilot commands. By contrast, here 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 techniques have been successfully used to verify the correctness of digital circuits, security, and communication protocols [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. 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.

3 Finding Software Vulnerabilities in UAVs Using Software Verification

3.1 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 (way-point mission engine), where the UAV will follow this route throughout its mission instead of using a traditional remote controller directly [9].

Dronekit[6] is an open-source software project, which allows one to command a UAV using Python script. In addition, 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 with the goal of gaining control of the UAV by means of the "vehicle" object. This control process relies on the planning software inside the UAVs system, which in some cases the embedded software might be permanently connected to the pilot controlling system (e.g., Remote Controller or Ground Control Station) due to live feedback or for real-time streaming.

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

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

Our main goal is to investigate in depth open-source UAVs code (e.g., DJI Tello[7] and Parrot Bebop[8]) for any potential security vulnerabilities. For example, Fig. 4 depicts 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 and then connects GCS to the UAV, by using the predefined port and IP address in lines 19 and 20. As we can see in line 22, 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. 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 operation of UAVs.

[6] https://github.com/dronekit/dronekit-python
[7] https://github.com/dji-sdk/Tello-Python
[8] https://github.com/amymcgovern/pyparrot

```
import socket
from time import sleep
import curses
INTERVAL = 0.2
def report(str):
    stdscr.addstr(0, 0, str)
    stdscr.refresh()
if __name__ == "__main__":
    stdscr = curses.initscr()
    curses.noecho()
    curses.cbreak()

    local_ip = ''
    local_port = 8890
    socket=socket.socket(socket.AF_INET,socket.SOCK_DGRAM)
    # socket for sending cmd
    socket.bind((local_ip, local_port))

    tello_ip = '192.168.10.1'
    tello_port = 8889
    tello_adderss = (tello_ip, tello_port)

    socket.sendto('command'.encode('utf-8'), tello_adderss)
    try:
        index = 0
        while True:
            index += 1
            response, ip = socket.recvfrom(1024)
            if response == 'ok':
                continue
            out = response.replace(';', ';\n')
            out = 'Tello State:\n' + out
            report(out)
            sleep(INTERVAL)
    except KeyboardInterrupt:
        curses.echo()
        curses.nocbreak()
        curses.endwin()
```

Fig. 4. Simple code to read and view the various data status of Tello UAV.

3.2 Illustrative Example – UAV swarm

Throughout this paper, we use an illustrative example from UAV swarm. In particular, we participated in a competitive-exercise[9], organized in early 2019 and with participants from across five different UK universities. The main goal of this event was teams from across the UK to compete against each other in a game of offense and defense 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 our modern society. This competition was sponsored by the British multinational defense, security, and aerospace company (BAE). Solutions 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.

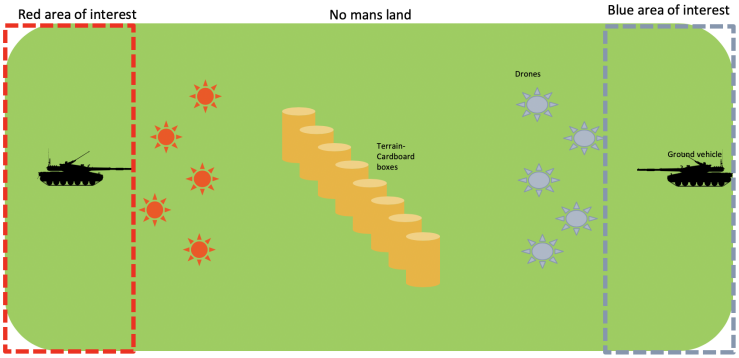

Fig. 5. UAV Swarm Competition.

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.

3.3 Insights about the combination of Fuzzing and BMC Approaches to UAV software

We describe our initial insight about a novel approach called "UAV Fuzzer" to investigate security vulnerabilities in UAVs (e.g., no buffer overflow, no dereferencing of null pointers, and no pointers pointing to unallocated memory regions). In particular, in order to detect security vulnerabilities in UAV software, we first

[9] `https://www.youtube.com/watch?v=dyyaY1VXqL4`

run a fuzzer engine using pre-collected test cases[10], with the goal of initially exploring the state-space of the UAV software operation. Here we keep track of the execution paths, which have been initially explored by our fuzzer; in particular, when the fuzzer engine gets stuck (e.g., the mutations generated were not suited enough to the new state transitions), our BMC tool runs against the target software to symbolically explore its uncovered state-space with the goal of checking the unexplored execution paths of the UAV software.

Our initial evaluation of the BMC approach relies on DepthK [25], which is k-induction verifier based on invariant inference for C programs. In particular, DepthK uses BMC techniques and invariant generators such as PIPS [23] and PAGAI [15], in order to both falsify and verify C programs. PIPS is an interprocedural source-to-source compiler framework for C and Fortran programs. PAGAI is a tool for automatic static analysis, which is able to automatically produce inductive invariants; both invariant generators rely on a polyhedral abstraction of program behavior for inferring invariants.

As a result, our UAV fuzzer is used to find any potential security vulnerabilities (e.g., buffer and integer overflow) and to address the issues BMC and fuzzing techniques still face when dealing with real-world UAV software, such as complex checks guarded by the execution paths of the program as well as deployment challenges. In addition, our UAV fuzzer is used to investigate whether it is possible to hijack the UAV through its embedded software. Successfully exploiting such security vulnerabilities will have a wider impact, because of the sensitivity of the UAV and the role embedded software plays on the UAV system. The main steps for our proposed verification algorithm are as below:

Algorithm 1 UAV Fuzzer

1: Define pre-collected test cases to be employed by the fuzzing engine.
2: Fuzzer engine begins to explore the first execution path in our UAV software and produce malformed inputs to test for potential security vulnerabilities.
3: Repeat step 2 until the fuzzer engine reaches a crashing point or it cannot explore the next compartment (due to complex guard checks).
4: DepthK runs the two invariant generators to produce inductive invariants with the goal of feeding them into a k-induction-based verification algorithm.
5: Guide DepthK to verify execution paths that have not been previously explored by our fuzzer engine in steps 2 and 3.
6: Repeat steps 4 and 5 until DepthK falsifies or verifies safety/security properties in UAVs.
7: Once the DepthK completely verifies the UAV code in step 6, it returns “false” if a property violation is found, “true” if it is able to prove correctness, or “unknown”.

[10] Note that a test case should be similar to a real valid data, but it must contain a problem on it, or also called “anomalies”. For example, to fuzz Microsoft office, a test case should be a word document or excel sheet, so the mutated version generated of such a similar package is called a test case.

3.4 UAV Communication Channel

An UAV has a radio to enable and facilitate remote communication between the ground control station 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, emerge as a priority in successful deployment of any UAV system.

A successful false-data injection attack, which had devastating effects on the UAV system was demonstrated by Strohmeier et al. [27], where 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, where Europe and US must comply with the mandatory use of this insecure protocol by the end of 2020.

In order 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. 6. 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.

4 Preliminary Experimental Evaluation

We have performed a preliminary evaluation of our proposed verification approach to detect security vulnerabilities in UAVs. In particular, 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.

4.1 Description of Benchmarks

The International Software Verification Competition (SV-COMP) [28], 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:

Fig. 6. *Spoofing*: GPS-satellite-signal is overlaid by a spoofed GPS-signal.

- *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 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. In order to analyze the radio frequencies and get in-depth sight of it, 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.

4.2 Objectives

The impact of our study is a novel insight on the UAV security potential risks. In summary, our evaluation has the following two 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?

4.3 Results

Our experiments demonstrate the effectiveness of the verification process of some of the vulnerabilities that are difficult to detect. Our experiments also show signs of the fragility of a commercial UAV system as described below.

SV-COMP. From a different angle, concurrency bugs in UAVs are one of the most difficult vulnerabilities to verify [26]. Our software verifier DepthK [25] 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[11] against other software verifiers. Our verifier showed promising results over thousands of verification tasks, which are of particular interest to UAVs security (*e.g.* *ConcurrencySafety* and *Overflows* categories), which answers **EQ1**. DepthK was able to compete against many other verifiers during SV-COMP 2019. *ConcurrencySafety* 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 1. 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. In addition, Fig. 7 and 8, extracted from SV-COMP 2019, show DepthK's results when compared with other verifiers.

Fuzzing Approach. According to a prior study [1], 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[12], which was chosen for evaluation purposes, are shown in Table 2. 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.

[11] `https://sv-comp.sosy-lab.org/2019/`
[12] `https://www.sumatrapdfreader.org/free-pdf-reader.html`

Table 1. 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

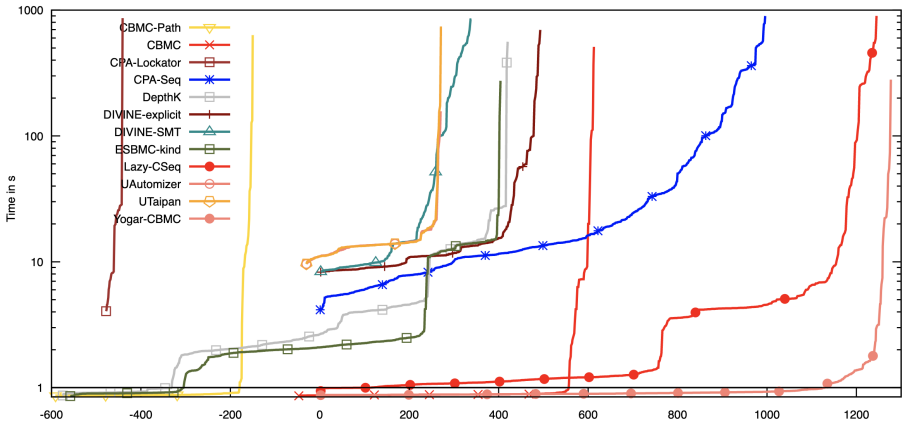

Fig. 7. Concurrency Safety category among other verifier tools [28].

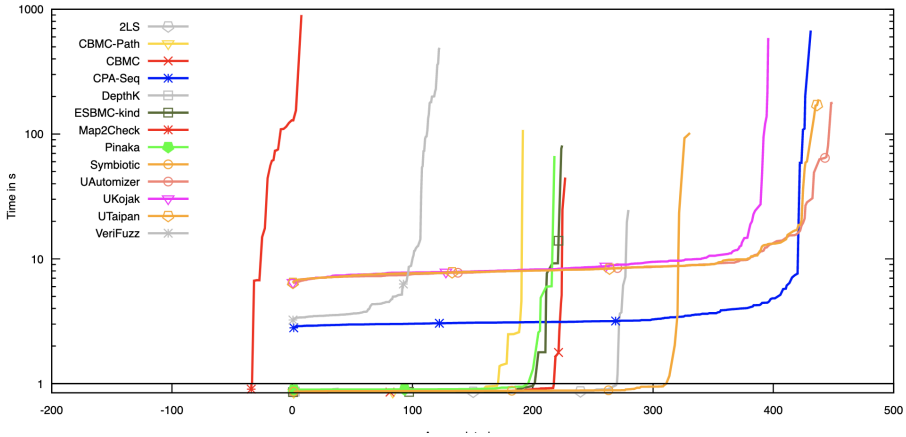

Fig. 8. Overflow category among other verifier tools [28].

Table 2. Fuzzing Approaches Comparison.

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

UAV Swarm Competition. As part of our participation at the UAV swarm competition sponsored by (BAE)[13], 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 resulted in managing to deliberately to crash UAVs or to take the control of different 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 3, where we describe the employed UAV models and tools and whether we were able to obtain full control or crash.

Table 3. Results of the UAV Swarm Competition.

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

4.4 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, 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).

5 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:

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

- **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.

References

1. Alhawi, O., Akinbi, A., Dehghantanha, A.: Evaluation and Application of Two Fuzzing Approaches for Security Testing of IoT Applications. In: Handbook of Big Data and IoT Security, pp. 301–327. Springer, Cham (2019).
2. Artzi, S., Kie zun, A., Dolby, J., Tip, F., Dig, D., Paradkar, A., Ernst, M.D.: Finding Bugs in Dynamic Web Applications. ISSTA pp. Pages 261–272 (2008).
3. Austin, R.: UNMANNED AIRCRAFT SYSTEMS UAVS DESIGN, DEVELOPMENT AND DEPLOYMENT. A John Wiley and Sons **54** (2011).
4. Bhatti, J.A., Shepard, D.P., Humphreys, T.E.: Civilian GPS Spoofing Detection based on DualReceiver Correlation of Military Signals . `file:///Users/lucasccordeiro/Downloads/dualCorr_psiaki.pdf` (2011), [Online; accessed 24-June-2019]
5. Biere, A.: Handbook Of Satisfiability, vol. 185, chap. 26. IOS Press (2009)
6. Chaves, L., Bessa, I., Ismail, H., Frutuoso, A., Cordeiro, L., de Lima Filho, E.: DSVerifier-Aided Verification Applied to Attitude Control Software in Unmanned Aerial Vehicles. IEEE Transactions on Reliability **67** (2018)
7. Cook, B., Khazem, K., Kroening, D., Tasiran, S., Tautschnig, M., Tuttle, M.R.: Model checking boot code from AWS data centers. In: CAV LNCS 10982, pp. 467–486. Springer (2018).
8. Corina S. Pasareanu: Using Symbolic (Java) PathFinder at NASA. Nasa (2010), `https://pdfs.semanticscholar.org/8933/ac2dbeb3ccd8cf3e393d8dbb22ccc4452b64.pdf`
9. Day, M.A., Clement, M.R., Russo, J.D., Davis, D., Chung, T.H.: Multi-uav software systems and simulation architecture. In: ICUAS. pp. 426–435 (2015).
10. Dey, V., Pudi, V., Chattopadhyay, A., Elovici, Y.: Security vulnerabilities of unmanned aerial vehicles and countermeasures: An experimental study. In: VLSI, pp. 398–403, (2018).
11. Farooq, M., Waseem, M., Khairi, A., Mazhar, S.: Article: A critical analysis on the security concerns of internet of things (iot). International Journal of Computer Applications **111**(7), 1–6 (February 2015).

12. Frei, S., May, M., Fiedler, U., Plattner, B.: Large-scale vulnerability analysis. In: LSAD. https://doi.org/10.1145/1162666.1162671
13. Gadelha, M.R., Monteiro, F.R., Morse, J., Cordeiro, L.C., Fischer, B., Nicole, D.A.: ESBMC 5.0: an industrial-strength C model checker. In: ASE. pp. 888–891. ACM Press (2018).
14. Godefroid, P., Levin, M.Y., Molnar, D.: Automated Whitebox Fuzz Testing. Queue - Networks (2012), https://www.eecs.northwestern.edu/{~}robby/courses/395-495-2017-winter/ndss2008.pdf
15. Henry, J., Monniaux, D., Moy, M.: PAGAI: A Path Sensitive Static Analyser. In: Electron. Notes Theor. Comput. Sci. pp. 15–25. Elsevier Science Publishers B. V. (2012)
16. Jamie Tarabay, O.L., Lee, I.: Israel: Iranian drone we shot down was based on captured US drone - CNN (2018), https://edition.cnn.com/2018/02/12/middleeast/israel-iran-drone-intl/
17. Javaid, A.Y., Sun, W., Devabhaktuni, V.K., Alam, M.: Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In: HST, pp. 585–590, (Nov 2012).
18. Joanna Frew: An overview of new armed drone operators The Next Generation. Tech. rep., Drone Wars UK, Oxford (2018), www.dronewars.net
19. Kerns, A.J., Shepard, D.P., Bhatti, J.A., Humphreys, T.E.: Unmanned aircraft capture and control via gps spoofing. Journal of Field Robotics **31**(4), 617–636 (2014).
20. Miller, B.P., Cooksey, G., Moore, F.: An empirical study of the robustness of macos applications using random testing. In: Proceedings of the 1st International Workshop on Random Testing, RT. pp. 46–54. ACM (2006)
21. Nassi, B., Ben-Netanel, R., Shamir, A., Elovici, Y.: Game of Drones-Detecting Streamed POI from Encrypted FPV Channel. In: MobiSys (2018), https://arxiv.org/pdf/1801.03074.pdf
22. Paques, M., Verbij, R.: EXPLORING SECURITY VULNERABILITIES OF UNMANNED AERIAL VEHICLES. Ph.D. thesis, University of Twente (2015), https://www.jbisa.nl/download/?id=17706129
23. ParisTech, M.: PIPS: Automatic Parallelizer and Code Transformation Framework. Available at http://pips4u.org (2013)
24. PwC: Drones could add £42bn to UK GDP by 2030 - PwC research (2018), https://www.pwc.co.uk/press-room/press-releases/pwc-uk-drones-report.html
25. Rocha, W., Rocha, H., Ismail, H., Cordeiro, L., Fischer, B.: Depthk: A k-induction verifier based on invariant inference for c programs. In: TACAS, LNCS 10206, pp. 360–364, (2017).
26. Sirigineedi, G., Tsourdos, A., Żbikowski, R., White, B.A.: Modelling and Verification of Multiple UAV Mission Using SMV. EPTCS **20**, 22–33 (2010).
27. Strohmeier, M., Lenders, V., Martinovic, I.: Intrusion detection for airborne communication using phy-layer information. In: Almgren, M., Gulisano, V., Maggi, F. (eds.) Detection of Intrusions and Malware, and Vulnerability Assessment. pp. 67–77. Springer International Publishing, Cham (2015)
28. Dirk Beyer: Automatic Verification of C and Java Programs: SV-COMP 2019. TACAS (3), LNCS 11429, pp. 133–155 (2019).