Finding Security Vulnerabilities in Unmanned Aerial Vehicles Using Software Verification

Joint work with Mustafa A. Mustafa and Lucas C. Cordeiro

Omar M. Alhawi
University of Manchester
omar.alhawi@Manchester.ac.uk
2019
Software is Everywhere
Exploitable Software is Everywhere

Security vulnerabilities can lead to drastic consequences

Attacked by rogue camera software and by a malware delivered through a compromised USB stick.

The attackers were able to fully control Bird H-6U.

A sailor on the U.S.S. Yorktown entered a 0 into a data field in a kitchen-inventory program.

The 0-input caused an overflow, which crashed all LAN consoles and miniature remote terminal units.

The Yorktown was non operational in the water for about two hours and 45 minutes.

https://www.boeing.com/defense/unmanned-little-bird-h-6u/
https://medium.com/@bishr_tabbaa/when-smart-ships-divide-by-zero-uss-yorktown-4e53837f75b2
• Unmanned Aerial Vehicles (UAVs) are systems-of-systems that couple their cyber and physical components.

Verifying Embedded Software in UAV is Hard Too

Increase in lines of code

HMI

real-time computer system (RTCS)

sensor

network

actuator

Machine learning

Mass production

multi-core processors with limited amount of energy

safety-critical applications
Security Challenges in UAVs

• Vulnerability analysis (software connected with hardware)
• Remote accessibility (device authentication, access control)
• Patch management (vendors might be long gone)
• Attacks from physical world (GPS spoofing and replay attack)
**Related Work**

*Literature in the area is scarce.*

- **Securing the MAVLink Protocol**[1]
  - MAVLink protocol, used for bidirectional communication between a drone and a ground control station.

- **Fuzzing the MAVLink protocol**[2]
  - Identify possible vulnerabilities in the protocol implementation using fuzzing technique.

---


Related Work

- **Smart Device Ground Control Station**[3]
  
  - Analyse the cyber security vulnerabilities within the communication links, smart devices hardware.

- **Autopilot systems**[4]
  
  - Identify the possible threats and vulnerabilities of the current autopilot system.

- **Existing Gaps:**
  
  - No software evaluation
  - No support to the drone's high-level layer
  - No specific functionality for verification decisions


[4]"Cyber Attack Vulnerabilities Analysis for Unmanned Aerial Vehicles’,. [Online]. Available: https://static1.squarespace.com/static/553e8918e4b0c79e77e09c4d/t/5ae86e6a8a922d40d2c0d1bd/1525182105346/AIAA-Infotech_Threats-and-Vulnerabilities-Analysis.pdf
Existing Gaps

• No software evaluation
• Malicious Software
• UAV software exploitation
• No support to the drone's high-level layer
• No specific functionality for verification decisions
Objectives

To design an effective approach to check UAV software implementations against vulnerabilities.

How vulnerable are the Drones to a cyberattack?

Develop a framework within which to think about and discussion cybersecurity in UAVs.
Project Approach

There are two main layers of drone programming.

1. Low level (Firmware):
Direct communication with the hardware being used, and provides the drone with its basic functionality.

2. High level (Software/Applications):
Treat your drone as a magical black box that reliably responds to commands send to it.

Our approach is to investigate the areas of UAV software vulnerabilities in order to improve software productivity.
RQ1: Are we able to perform successful cyber-attacks in commercial UAVs?
1- GPS Spoofing Attack

Spoofer

Transmit the TIME SIGNAL

Antenna that operates at 1575.42 MHz (L1 GPS) signal

OS, BladeRFx40

gps-sdr-sim -l <lat,long,alt> -d <duration>

Software Defined Radio (SDR)

Transform the IQ data into RF output

Detect

Transmit the TIME SIGNAL

Tello & Bebop 2.. UAV

Disrupted Path

2- Denial of Service

Spoofing

Track
# Results

Results from UAV Swarm Competition

<table>
<thead>
<tr>
<th>Vulnerability type</th>
<th>Drone Model</th>
<th>Tool</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spoofing</td>
<td>Parrot bebop 2</td>
<td>Wi-Fi transmitter</td>
<td>Full Control</td>
</tr>
<tr>
<td>Denial of service</td>
<td></td>
<td></td>
<td>Crash</td>
</tr>
<tr>
<td>Spoofing</td>
<td>Tello</td>
<td>Wi-Fi transmitter</td>
<td>Full Control</td>
</tr>
<tr>
<td>Denial of service</td>
<td></td>
<td></td>
<td>Full Control</td>
</tr>
</tbody>
</table>

DepthK: K-Induction + Invariant Inference

DepthK employs **Bounded Model Checking** (BMC) and **k-Induction** based on program invariants, which are automatically generated using **polyhedral constraints**

- DepthK uses ESBMC, a context-bounded symbolic model checker that verifies single- and multi-threaded C programs

- DepthK uses PAGAI and PIPS tools to infer program invariants
DepthK: K-Induction + Invariant Inference

```c
int x = 0;
int t = 0;
int phase = 0;
while ( t < 100 ) {
    if ( phase == 0 )
        x = x + 2;
    if ( phase == 1 )
        x = x - 1;
    phase = 1 - phase;
    ++t;
}
assert( t <= 100 );
```
Experimental Questions

- Supporting fuzzing, BMC, and analysis of UAV’s software.

- RQ2: Can DepthK help us understand the security vulnerabilities that have been detected?
Results from Software Verification competition  
SV-Comp19

<table>
<thead>
<tr>
<th>Category</th>
<th>Benchmarks</th>
<th>Correct Results</th>
<th>Incorrect Results</th>
<th>Unknown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concurrency Safety</td>
<td>1082</td>
<td>966</td>
<td>20</td>
<td>96</td>
</tr>
<tr>
<td>No Overflows</td>
<td>359</td>
<td>167</td>
<td>0</td>
<td>192</td>
</tr>
</tbody>
</table>

Experimental Questions

- Supporting fuzzing, BMC, and analysis of UAV’s software.

• RQ3: Can generational or mutational fuzzers be further developed to detect vulnerabilities in real-world software?
How the data input (test cases) used during fuzzing process influence the fuzzing result?

Mutators depend on the input they are modifying.

Test programs on random unexpected data can be quite effective. Can be realized using black/white.

Usually implemented via instrumentation.

Tricky to scale for programs with many paths.
UAV Fuzzer Framework

Fuzzer Test Case

```python
while True:
    index %= 1
    if + replaced with %
    response, ip = socket.recvfrom(1024)
    if response == 'ok'
        continue
```

Model Checking

All the sequences after fuzzing engine stuck will symbolically Executed to determine if they can reach an exploitation primitive.
UAV Fuzzer Framework (cont.)

1. Classify input variables into symbolic and/or concrete
2. Instrument to record symbolic vars and path conditions
3. Choose an arbitrary input
4. Execute the program
5. Symbolically re-execute the program
6. Negate the unexplored last path condition
Challenges

• Benchmark selection.

• The size of complex software implementations.

• Scaling Issues for Symbolic Exploration.

• Time required.
Methodology and Evaluation

- Our proposed approach, “UAV Fuzzer” Can be evaluated in three aspects:

  - The quality of the test cases
    - Code coverage achieved
  - Bugs detection
    - Validating UAV software implementations
  - The verification time
    - Results comparison
Contributions

- The contribution of this research are as follows:

  - Provide: A better understanding of fuzzing and BMC.
  - Identify: UAV vulnerabilities.
  - Detect: Vulnerabilities in UAV Software.
  - Employ: UAV fuzzer for a software exploration.
  - Use: BMC and Fuzzing to generate high coverage.
  - Compare: With other software verifiers and fuzzers.
Research Mission

Automated verification to ensure the software security in UAVs

Methods, algorithms, and tools to write software with respect to security
QUESTIONS?

omar.alhawi@Manchester.ac.uk