MANCHESTER 1824
The University of Manchester

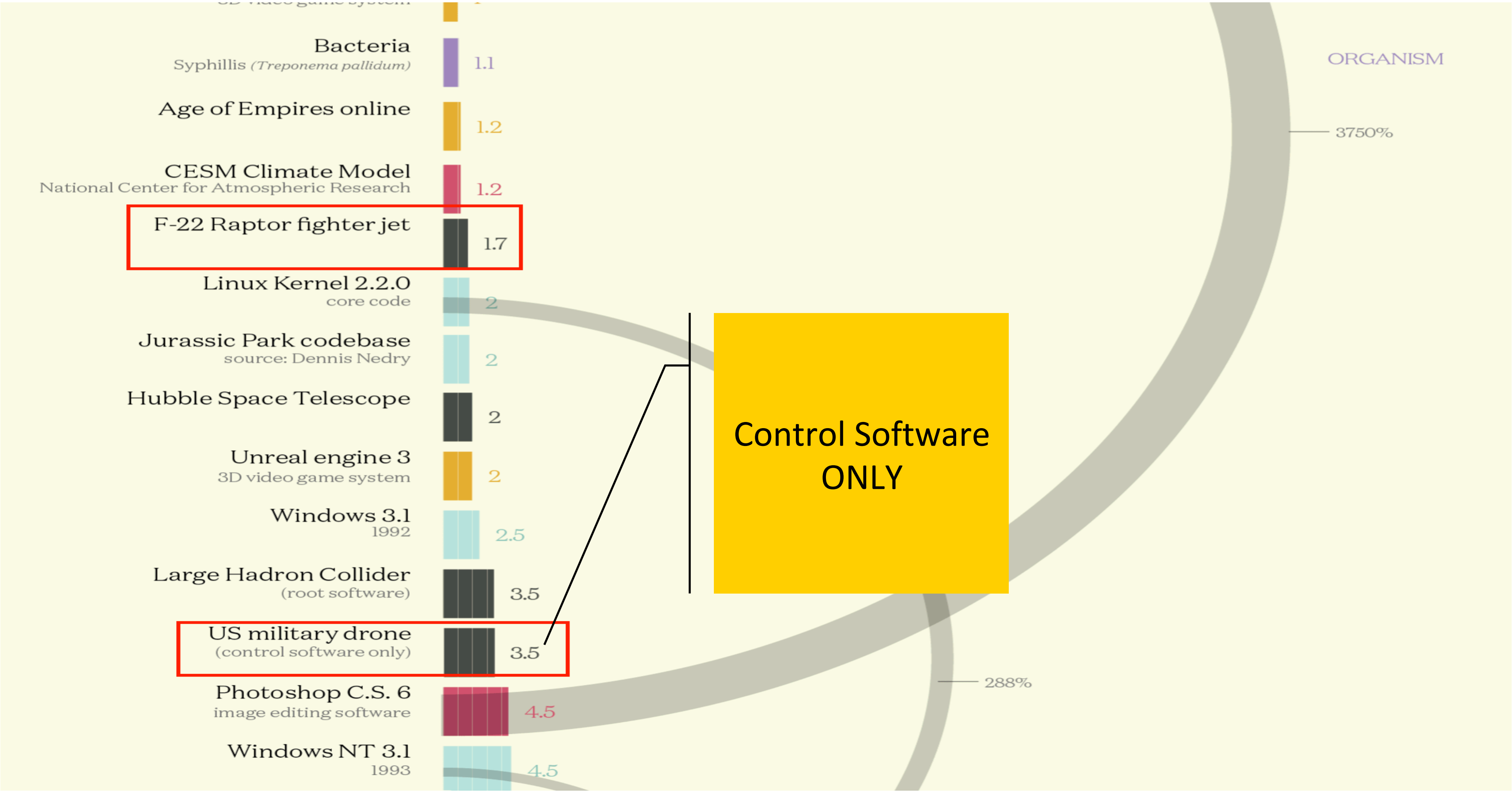
Finding Security Vulnerabilities in Unmanned Aerial Vehicles Using Software Verification

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

Software is Complex



https://informationisbeautiful.net/visualizations/million-lines-of-code/

Exploitable Software is Everywhere

Security vulnerabilities can lead to **drastic consequences**

Boeing Unmanned Little Bird H-6U

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 .

USS Yorktown aircraft carriers

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-zer0-uss-yorktown-4e53837f75b2

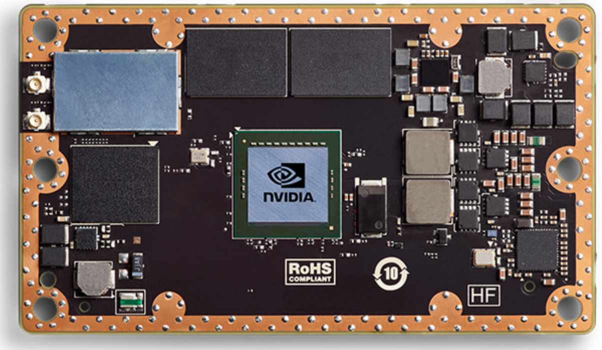
Verifying Embedded Software in UAV is Hard Too

- Unmanned Aerial Vehicles (UAVs) are **systems-of-systems** that couple their **cyber** and **physical** components

Increase in lines of code

Machine learning

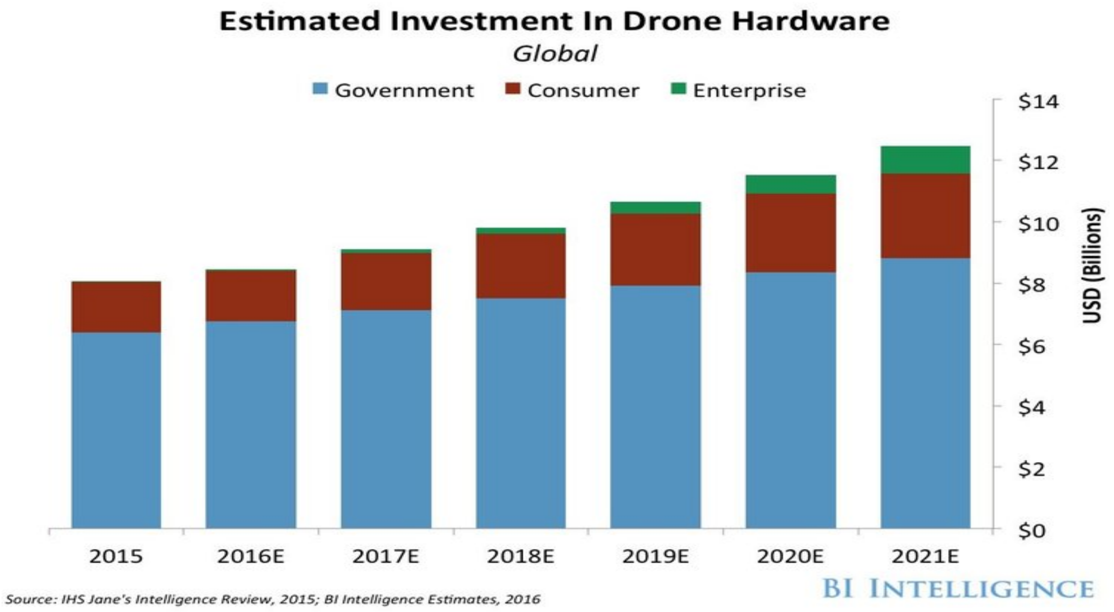
Mass production



[multi-core processors with limited amount of energy]

safety-critical applications

Security Challenges in UAVs



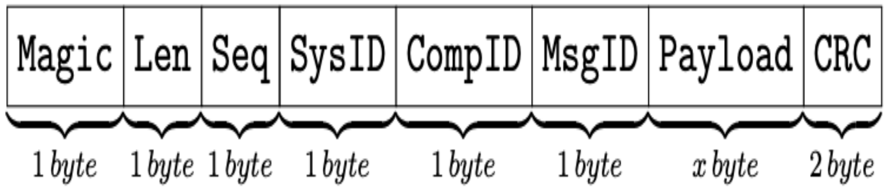
Cyberwarfare

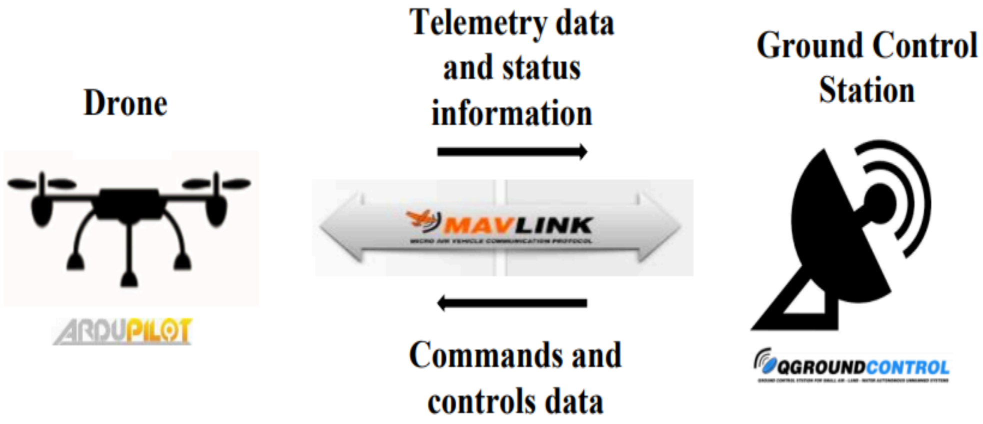
Cyberterrorism

Cyberhooliganism

- 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.

[1] "MAVSec: Securing the MAVLink Protocol for Ardupilot/PX4 Unmanned Aerial Systems', 2019. [Online]. Available: https://ieeexplore.ieee.org/document/8766667

[2] "Security Analysis of the Drone Communication Protocol: Fuzzing the MAVLink protocol, 2016 [Online]. Available: https://www.esat.kuleuven.be/cosic/publications/article-2667.pdf

The University of Manchester

MANCHESTER 1824

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

[3]"Unmanned Aerial Vehicle Smart Device Ground Control Station Cyber Security Threat Model '. [Online]. Available: https://ieeexplore.ieee.org/document/6699093

[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.

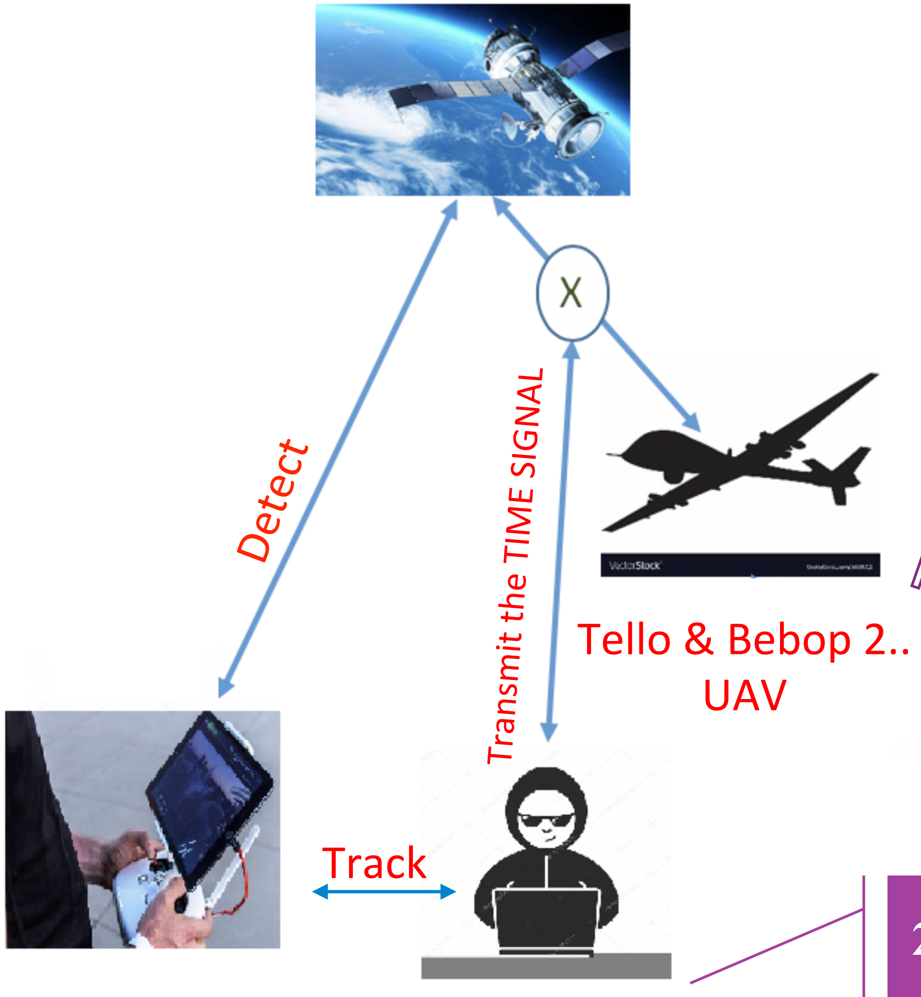
2. High level (Software/Applications):
Treat your drone as a magical black box that reliably responds to commands send to it.

Our approach is to investigate the areas of UAV software vulnerabilities in order to improve software productivity.

Experimental Question

RQ1: Are we able to perform successful cyber-attacks in commercial UAVs?

1- GPS Spoofing Attack



Software Defined Radio (SDR)

Transform the IQ data into RF output

Antenna that operates at 1575.42 MHz (L1 GPS) signal

OS, BladeRFx40

Disrupted Path

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

2- Denial of Service

Results

Results from UAV Swarm Competition

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

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

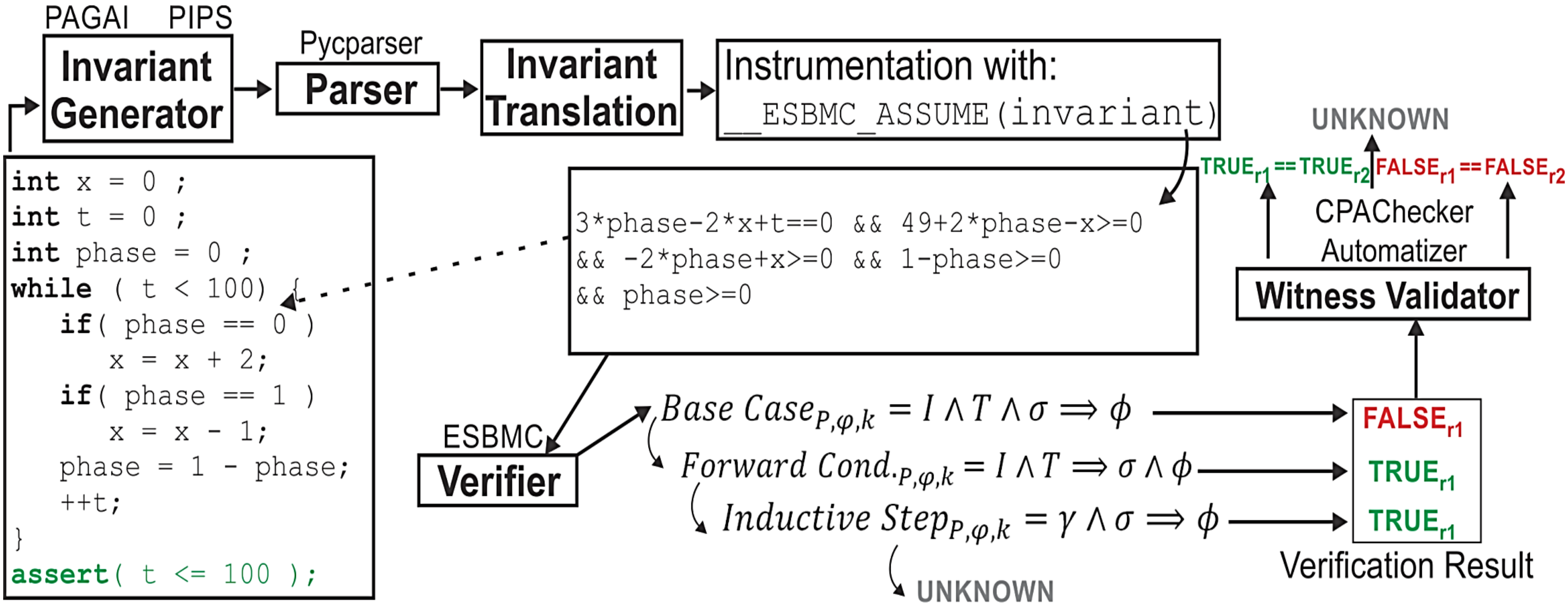
DepthK: K-Induction + Invariant Inference

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



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

Category	Benchmarks	Correct Results	Incorrect Results	Unknown
Concurrency Safety	1082	966	20	96
No Overflows	359	167	0	192

https://sv-comp.sosy-lab.org/2019/committee.php

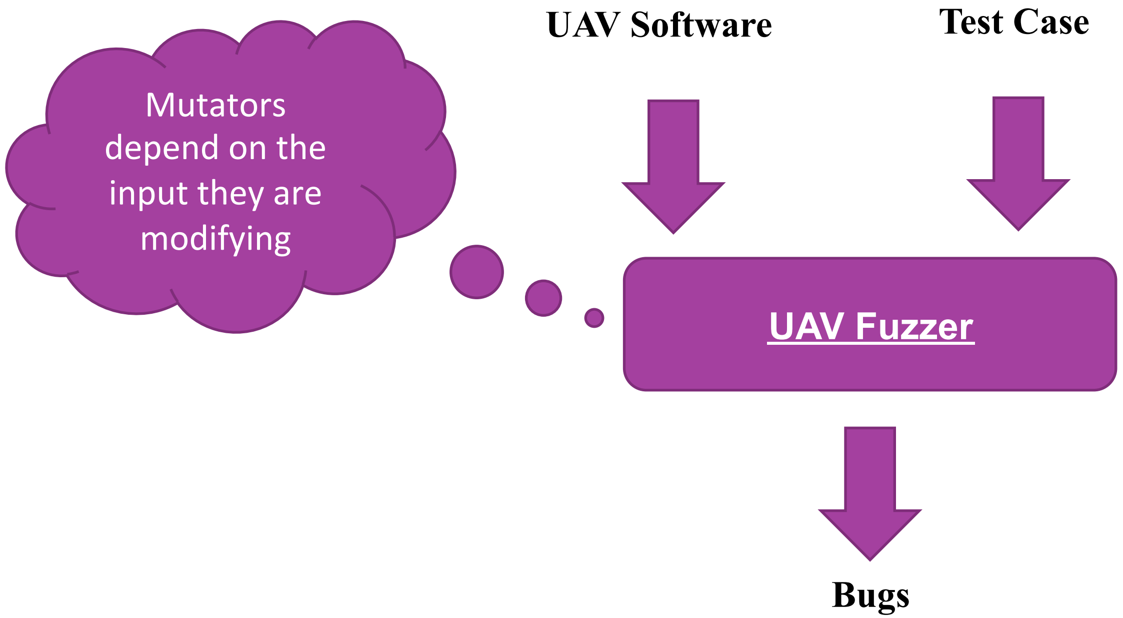
Experimental Questions

➢**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?

BMC& Fuzzing
Representation

Future Work: UAV Fuzzer Framework



- Test programs on random unexpected data
- Can be realized using black/white
- Can be quite effective
- Usually implemented via instrumentation
- Tricky to scale for programs with many paths

How the data input (test cases) used during fuzzing process influence the fuzzing result**?**

UAV Fuzzer Framework

Fuzzer Test Case

```
while True:
  index %= 1
# + 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.

Read and view Tello UAV data status

```python
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:
```

UAV Fuzzer Framework (cont.)

Classify input variables into symbolic and/or concrete

↓

Instrument to record symbolic vars and path conditions

↓

Choose an arbitrary input

↓

Execute the program

↓

Symbolically re-execute the program

↓

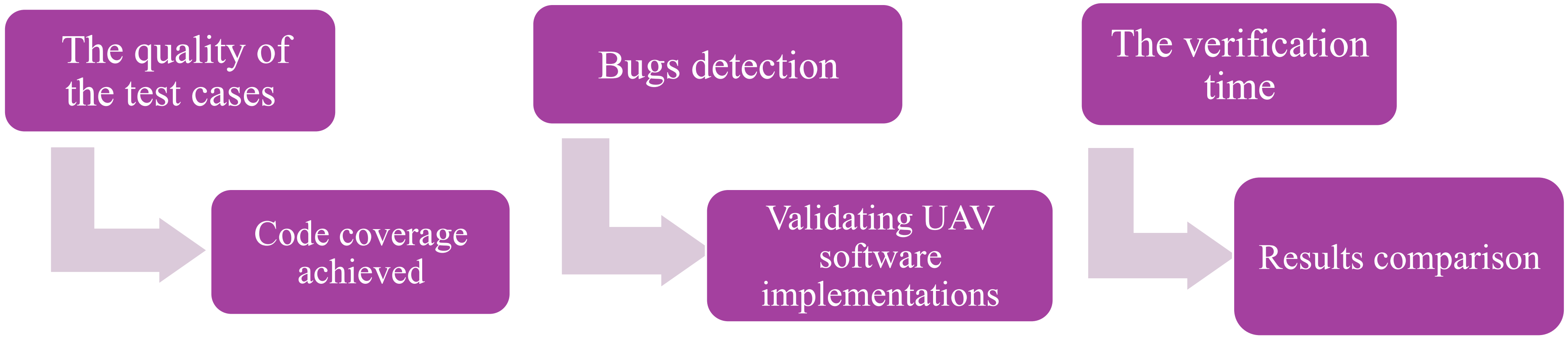
Negate the unexplored last path condition

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:



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