ATTACK ON THE DRONES

Oleg Petrovsky
HP Security Research, USA

Email oleg.petrovsky@hp.com

ABSTRACT

In this paper we will analyse various popular multi-rotor unmanned aerial vehicle (UAV) configurations and controllers for susceptibility to known and proof-of-concept security attacks.

The study will include analysis of existing malicious attack claims and their validity when applied to the world's leading open-source UAV controllers, such as *3D Robotics' ArduPilotMega* (*APM*), the joint project of *3D Robotics'* and *PX4 groups*' *Pixhawk*, and similar popular flight controllers recognized in the UAV industry.

The paper will examine the attack surface of existing UAV platforms as exhibited by a UAV's different functional modules – software, firmware, hardware and controls – as well as those stemming from the environment.

Possible future attack scenarios as well as possible ways of hardening against such attacks are considered.

INTRODUCTION

The explosive growth in popularity of multi-rotor unmanned aerial vehicles (UAVs) can be attributed to advances in inertial-measurement sensors, microcontrollers and battery technologies, all of which stem from research spent on development of personal devices such as mobile phones and laptops. It is not a secret that the helicopter and multi-rotor technology, as well as fixed-wing unmanned aerial vehicle technology, exists and has been in development for some time, albeit at a somewhat slow pace. The complexity of controlling an aerial vehicle and the expense of development and manufacturing kept this technology away from a mass market for a time.

The situation changed when the technology for manufacturing and simplifying the operation of UAVs became accessible and inexpensive enough to become ubiquitous. The industry is estimated to grow to $11 billion a year by 2024 [1].

SCOPE AND APPLICATIONS

There are already a number of ambitious projects that aim to tap into the emerging UAV market. The most famous of these are known as *Amazon Prime Air* [2] and the recently abandoned *Google X*'s *Project Wing* [3]. These efforts are both aimed at shortening the delivery time for receiving goods. Beyond these well-publicized research cases (which haven't made it out of the lab environment just yet), it is probably less well known that multi-rotors are already becoming mainstream, and being used to deliver goods to remote areas – and, in some cases, making pizza available fast and fresh to a neighbour near you [4].

What does it mean for the rest of us? Chances are, it means that in the next few years multi-rotor UAVs might occupy airspace over our heads and important infrastructures. With this in mind, we want to assess the security levels of multi-rotors and examine possible UAV attack surfaces from the prospective of an attacker.

BASICS OF MULTI-ROTOR DESIGN

The majority of multi-rotor drones adhere to the same design frame. Its basic blocks, shown in Figure 1, are as follows:

A flight controller (FC) consists of a number of sensors and an embedded processing unit, normally an 8- or 32-bit system on a chip (SoC) that can be coupled with an external flash memory.

The block of sensors consists of a variety of inertial measurement unit (IMU) devices, which are responsible for a six degrees of freedom (DoF) aerial vehicle in spatial orientation, plus a magnetic orientation sensor, a pressure sensor, and a GPS unit, which contribute another five DoF. In total, a flight controller can rely on a 11 DoF sensor unit. The IMU sensors group usually includes two three-directional micro-electro-mechanical systems (MEMS) – an

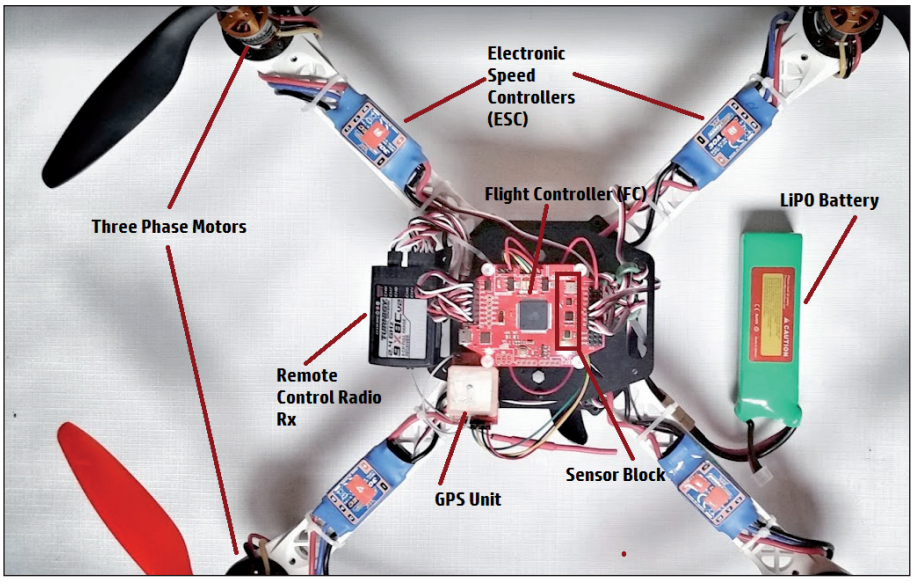


Figure 1: Basic components of a multi-rotor aerial vehicle.

accelerometer and a gyroscope. Often, the IMU group is housed in the same chip, which reduces electrical noise and raises its sensitivity. The sensors interface with the embedded CPU using a low-bandwidth serial protocol such as I2C, SPI, or UART.

The convenience of daisy-chaining the sensors and using only two lines for communications highlights the I2C protocol as one of the preferable choices.

The flight controller is in turn connected to the radio receiver, the electronic speed controller (ESC) units, and the power distribution board. The ESC units are each connected to an electric motor and the power distribution board. The flight controller can be also connected to a telemetry radio unit as well as to an on-screen display unit (OSD). The low-speed serial protocols, such as UART, SPI and CAN, are used. The ESC units are controlled by means of pulse width modulation and translate the control signal from the embedded CPU to the speed of an electric motor. The electric motors provide thrust to an aerial vehicle.

The entire system is controlled via a feedback loop, parameters of which include sensor modules, GPS, the radio receiver, and the telemetry data link inputs. This control loop conforms to a proportional-integral-derivative mechanism (PID) [5]. The PID method of automation is often found in industrial control systems and is aimed at reducing error between measured and set point values. The entire mechanics of automated control and stable flight is quite complex and well beyond the scope of this paper.

Instead, we intend to examine possible points of attack on such systems within existing multi-rotor flight controller models. Many such configurations are used in consumer-grade multi-rotors and fixed-wing remote UAVs; the underlying technology differs little from board to board and even from manufacturer to manufacturer. This interoperability is a boon for development, but provides a homogenous surface (and a potentially great return on effort invested) for would-be attackers.

POPULAR FLIGHT CONTROLLERS

The range of popular flight controllers includes the following. The list is by no means complete, but the market being as it is, one could reasonably expect potential malware writers to be keenly aware of these commonly used components.

KK

The initial version of the *KK* flight controller was designed by Rolf R. Bakke, who is also known as KapteinKuk throughout the numerous UAV forums [6]. The controller began as a robust, no-frills controller with basic features – one that didn't require the use of a personal computer to configure the flight controller board. (Normally, one would connect the flight controller board and run special software to set and store parameters on it.) It relied on *Atmel*'s *ATmega48/88/168-AU* chip, which could be programmed through the In System Programming (ISP) header with an AVR-capable programmer such as *AVRISP mkII* or *STK500*, both also from *Atmel Corporation*.

The basic sensor set-up would include *ENC-03RC* piezoelectric gyroscopes by *Murata*. The analog signal proportional to the angle rate would be read and processed by a microcontroller, which would be internally clocked at 8MHz, and the proportional (P) and integral (I) settings for automated control could be tuned by adjusting potentiometers on the board, as can be seen in Figure 2. The board did evolve: the latest hardware version, *KK2.1*, has a more powerful microcontroller (the *ATmega644PA* by *Atmel*), and the sensor block has been replaced by the single-chip digital micro electro mechanical system (MEMS) inertial measurement unit (IMU) *6050* by *InvenSense*. The added LCD screen and control buttons allow more programmatic control and setup of the board without the use of a PC. The firmware can still be compiled and updated over the ISP header. The board, due to its in-field set-up simplicity, is still quite popular and is actively supported throughout the UAV community. The controller is open-source and open hardware. The firmware updates can be downloaded online [7] and flashed to the board through the ISP header using the ISP protocol by *Atmel* [8].

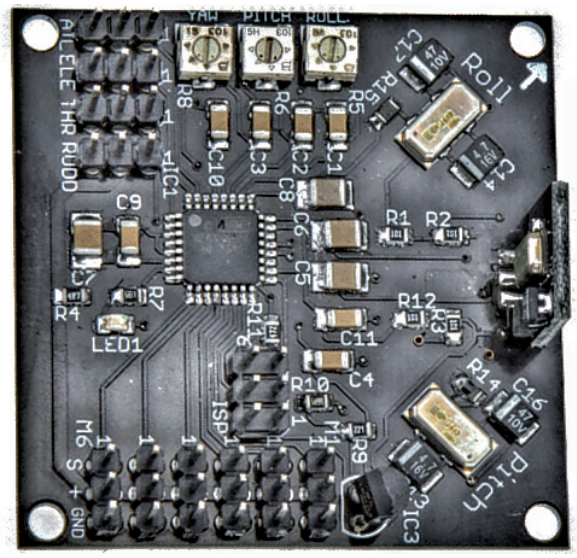


Figure 2: One of the numerous early KK clones.

MultiWii

MultiWii is yet another popular flight controller utilizing open source under GNU GPL v3 and open hardware. Its popularity can be attributed to its relative simplicity and the ability of the firmware to support a multitude of sensors and core CPUs [9]. The firmware is versatile enough to include an optional global positioning system (GPS) unit and an LCD for in-field tuning when away from a PC. The name of the initial versions of the firmware and hardware pays homage to the *Nintendo Wii* game console; earlier versions of the firmware relied on sensors found in the *Nintendo Wii Nunchuck*, the prevalence of which made the sensors cheap and easily obtainable. The *MultiWii* firmware was originally written for 8-bit *Atmel* microcontrollers using the processing language in the *Arduino* framework [10], but since then has successfully been forked and ported to support 32-bit processors such as the *STM32 CortexM3* series by *STMicroelectronics*. The 32-bit port of *MultiWii* is called '*BaseFlight*' [11].

Currently, popular 8-bit versions of the *MultiWii* capable hardware include the *MultiWii Pro*, *MultiWii Pro V2* and *CRIUS Pro V2*. Theses boards rely on the very capable *ATmega2560* 8-bit processor clocked at 16MHz with 256K of

Flash, 4K of EEPROM and 8K of RAM. On the 32-bit side are *Naze32* and its clones such as *Flip32*, *AfroFlight32* and others based on the *STM32F103CxT6 CortexM3 Core ARM* processor by *STMicroelectronics* (clocked at 72MHz, with Flash up to 128K and static RAM up to 20K). There is also a fork of the original '*BaseFlight*' called '*CleanFlight*', which is aimed at supporting an even greater number of existing 32-bit boards, such as *CC3D* by *OpenPilot* and *Sparky* by *TauLabs*. The '*CleanFlight*' fork [12] also supports *STMicroelectronics*' *STM32F3* and *STM32F1* processor targets.

In a sensor block, most of the 8- or 32-bit flight controller boards rely on the *MPU6050* by *InvenSense*, which houses a three-axis accelerometer, a three-axis gyroscope, *Honeywell*'s *HMC5883L* multi-chip module as a 3D magnetometer, and *MEAS Specialties*' *MS5611* or *Bosch*'s *BMP085* as a barometric pressure sensor, all as shown on the *MultiWii Pro V2* board in Figure 3. The sensor block chipset interfaces are digital and use I2C or SPI protocols, and all the analog-to-digital conversion and processing happens inside the sensor chips. This greatly reduces the signal noise caused by interferences and increases the quality and the speed of measurements.

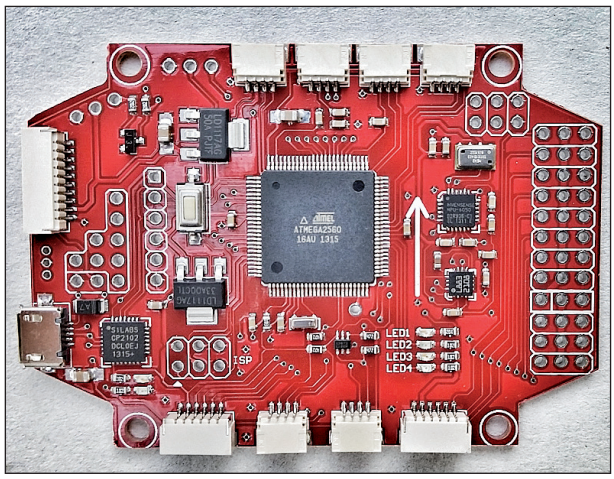


Figure 3: MultiWii Pro V2 flight controller.

The *MultiWii*-supported boards can be configured and controlled through changes in the firmware source header files, as well as through a number of GUI configuration programs running on a PC and interfacing with the FC board through the COM interface. The most popular configuration program is *MultiWiiGui_conf*, which is written in Java [13], and *MultiWii WinGUI* [14]. The 32-bit 'BaseFlight' builds use the cross-platform *Chrome Baseflight* configurator tool [15] and the 'CleanFlight' fork uses *Cleanflight Configurator* [16]. The GUI configuration programs communicate with the boards through a COM port interface using the *MultiWii* lightweight protocol [17]. The configuration programs are capable of making critical changes to the stable flight parameters as well as upgrading firmware using onboard boot-loader protocols.

OpenPilot

OpenPilot is yet another community effort aimed at developing hardware and software for multi-rotor UAV and fixed-wing planes [18]. The project was founded by David Ankers, Angus Peart and Vassilis Varveropoulos in late 2009. The hardware developments were concentrated around *STMicroelectronics*' *STM32F1* and recently moved to that company's *STM32F4* series of microcontrollers. Two popular flight controllers by *OpenPilot* are the *CC3D* and the *Revolution* (sometimes called the *Revo*) [19]. In 2010, the project was forked to support a wider variety of *STM32F3*- and *STM32F4*-based platforms as well as the different IMU selections. The fork is currently maintained independently by *TauLabs* [20]. There are build targets that include *Discovery*-series development boards from *STMicroelectronics*, such as *Discovery F3* and *Discovery F4*.

As part of this research and my UAV platform experiments, I designed and built a shield for the *Discovery F3* board conforming to the specifications of the *TauLabs* project, as shown in Figure 4.

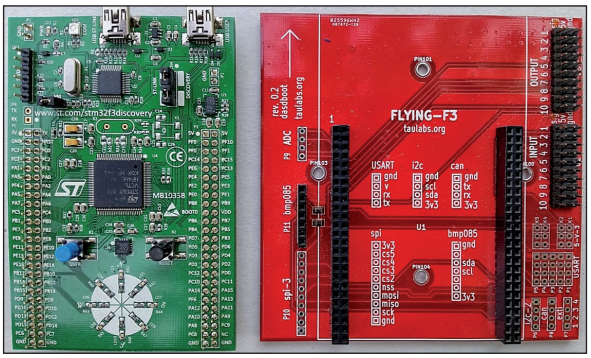
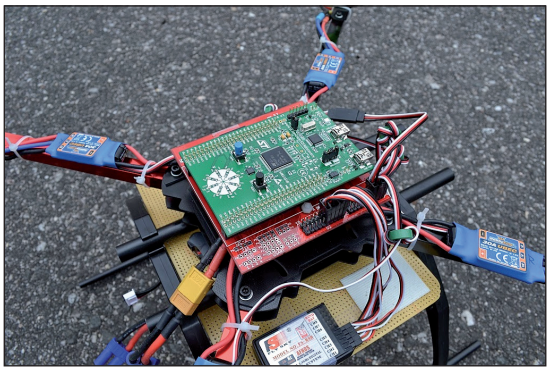
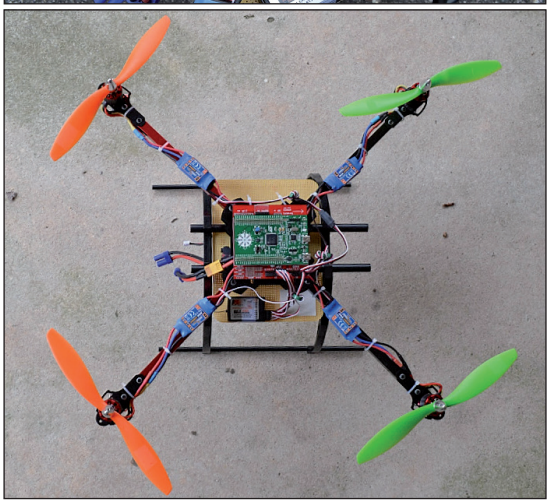


Figure 4: The Discovery F3 board and the Flying F3 shield.





Figures 5 and 6: Discovery F3-based quadcopter with TauLabs firmware; the varied propeller colours help the operator to discern the front and back of the UAV while it is aloft and moving in the line of sight.

This project allowed me to study the build process of the firmware and the upload process of the firmware to the target board, and to become familiar with the use of the *OpenPilot* telemetry protocol and the *OpenPilot* ground control station software. The board was successfully wired as shown in Figures 5 and 6, and tested in flight.

APM

This flight controller's history began in 2007, when it was envisioned by Chris Anderson over a weekend in an attempt to make an autopilot for a plane made out of the *Lego Mindstorm* set. In 2009, Chris Anderson and Jordi Munoz founded *3DRobotics*. Later the same year, Jordi Munoz created the *ArduPilot* code repository. In 2010, *ArduPilot* merged with *AeroQuads* to extend *ArduPilot* to support multi-rotor platforms. *ArduCopter* was born.

Since then the *ArduCopter* code and hardware have had a steady stream of development and advances. *3DRobotics*, a leading manufacturer of the *ArduCopter* boards, managed to find a good balance between commercializing a product and keeping the hardware designs and code open, while sponsoring a large *DIYDrones* community [21] and welcoming its cooperation and innovation [22]. (In this writer's opinion they are UAV open-source industry leaders and can be compared in the UAV world to *Linux* in the world of operating systems.) Currently, there are two flagship models: the *APM 2.6*, an 8-bit flight controller based on *Atmel*'s *ATmega2560*, and *Pixhawk*, based on the *PX4* open-hardware project and the *STMicroelectronics STM32F4* and *STM32F1* dual, 32-bit-based processor boards. The 8-bit controller relies on the *Arduino* framework and the firmware code is written in the Processing language. The firmware is uploaded using the *STK500* bootloader protocol over a serial interface.

The sensors module relies on the *InvenSense MPU6050* three-axis accelerometer and gyroscope as well as the previously mentioned three-axis *HMC5883L* magnetometer and *MS5611* barometer. Optionally, the board can make use of a GPS module such as *NEO-6*, *LEA-6*, *NEO-7* or *NEO-M8*, all manufactured by *u-blox*. *U-blox NEO-7* and *NEO-M8* are Glonass-capable and particularly useful in Europe. The GPS is connected to the board over a serial link and the data exchange is carried over using the UBX [23] protocol. UBX is a protocol proprietary to *u-blox*; it is designed to transmit GPS data to a host over an asynchronous serial link. The protocol is more compact than others, using 8-bit binary data as opposed to NMEA (which is ASCII-based and thus larger). It also uses a checksum algorithm that is two bytes lower in overhead, and a modular two-stage (Class and Message ID) message identifier. (Those readers familiar with RFC 1145 will recognize the checksum algorithm as the 8-bit Fletcher algorithm [24].)

Because of steady development and improvements in the flight controller features, the complexity of the firmware started testing the boundaries of the 8-bit board. According to DIYDrones [25], the *3DRobotics* team reached the maximum potential for the 8-bit flight controller and it looks like the newer versions of *ArduCopter* will not be running on the 8-bit *APM 2.6*.

The 32-bit *Pixhawk* board [26] is derived from the *PX4* project, which is further developed and supported by the Computer Vision and Geometry Lab, the Autonomous System Lab, and the Automatic Control Laboratory, all of ETH Zurich (Eidgenössische Technische Hochschule Zürich), the Swiss Federal Institute of Technology.

The *Pixhawk* was developed in collaboration with *3DR* with the *PX4* group and is aimed at reducing the cost of production and increasing the board's availability. The hardware design is open and several clones already exist on the market. The *Pixhawk* board is quite an improvement over the *APM* 8-bit flight controller; it is based on the *STM32F4 Cortex M4* series CPU and has a second *STM32F1* CPU as a failsafe option. In a sensor module, the *Pixhawk* flight controller can optionally rely on a combination of the *MPU6000* three-axis accelerometer and three-axis gyroscope by *InvenSense*, the 14-bit *STM LSM303D* accelerometer and magnetometer, the *STM L3GD20* three-axis 16-bit gyroscope [27], and the *MS5611* barometer. All of the above-mentioned options were present on the *Pixhawk* board clone with which the author was experimenting, as shown in Figure 7.

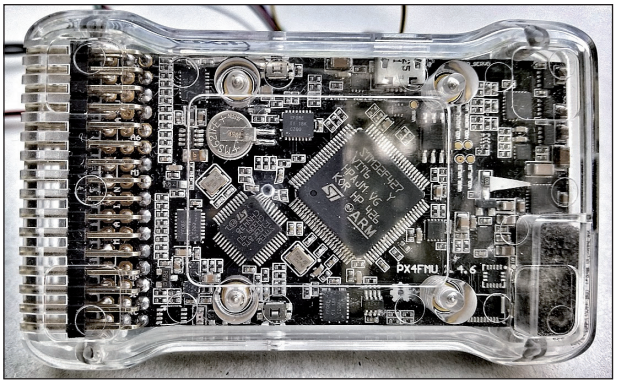


Figure 7: A Pixhawk board clone.

The board also can use an external three-axis magnetometer such as *HMC5883L*, and connects to a *u-blox* series GPS over a serial link.

The *Pixhawk* runs a multi-threaded real-time operating system (RTOS) called *NuttX* [28], which provides a POSIX-like environment. The software can be updated through a USB bootloader. *ArduCopter* has successfully been ported to run on *NuttX* as a multi-threaded application according to Andrew Tridgell, who is a lead developer of *ArduPilot* [29]. It runs in four threads and uses 'soft interrupt tasks' for sensor drivers. The firmware uses a hardware abstraction layer (HAL), which further simplifies porting *ArduCopter* to other platforms such as those based on *Linux*.

The configuration and firmware updates, as well as the handling of flight data by the *Pixhawk* and *APM2.6* boards, can be done through a number of GUI front ends, sometimes referred to as ground control stations (GCS) – for instance, an *APM Planner* [30] or *Mission Planner* [31].

The latest versions of *ArduCopter* firmware can be found on the *DIYDrones* site [32].

Linux-based UAV microcontrollers

In recent years, a number of cheap yet powerful *ARM*-based microcontrollers have appeared on the market, driven by the mobile devices industry. This led to production of inexpensive but quite computationally efficient embedded systems such as *RaspberryPI*, *RaspberryPI 2*, *Beaglebone* and others. These

boards are capable of running *Linux* and have enough raw input/output pins to control and acquire information from embedded electronic devices and sensors. These features, and the fact that *Linux* is well-established and rich with development tools for the platform, made it appealing to use such boards as flight controller main processing modules. What's left is to add the sensor block and port an open-source flight controller firmware. The *NavIO+* project has done just that [33]. It runs *APM* software, uses *MavLink* for communications, and supports a wide variety of ground control stations. The platform is open and can be used for development and research.

DJI Naza-M

The *Naza* by *DJI*, shown in Figure 8, is an example of a closed-source and hardware flight controller system. The sensor module is tightly housed and is not accessible for detailed viewing, the markings on the chips are not readable, and the CPU is in ball grid array (BGA) packaging with no protruding legs, which makes it difficult to connect to for reverse-engineering purposes.

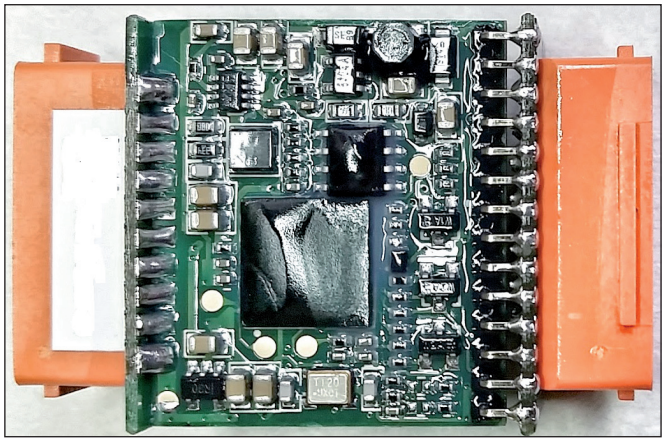


Figure 8: The closed-source DJI Naza flight controller.

The *DJI Innovations Science and Technology Company* started in 2006 and has its headquarters in Shenzhen, China [34]. *DJI Innovations* positions itself as a leader in manufacturing commercial and recreational unmanned air vehicles. Among other products, it is known as the company behind the *Phantom* and *Inspire* lines of quadcopters [35]. The company enjoyed a good run as virtually the only established manufacturer of recreational UAVs in the beginning, but currently it must compete with other companies as well as with the open-source and hardware community. At the lower price range in its production portfolio, *DJI* has a number of flight controllers such as the *Naza-M Lite* and *Naza-M V2*. According to various sources, the controllers seem to be based on the same or very similar hardware platforms, but are locked to particular firmware updates and assistant (configuration) software [36].

Most of the additional modules necessary for UAV building, such as GPS, LED status indicators, USB ports, power management units, and so forth, have to be bought from *DJI*. These modules are specific to a particular line of flight controllers. These restrictions are not really welcomed in the hobbyist community, and there have been numerous attempts to reverse-engineer protocols and use other commonly available modules with the *DJI* flight controllers [37–39].

There's also an unofficial way to upgrade firmware in *Naza-M Lite* to make it comparable in performance to *Naza-M V2* [40].

SimonK ESC

As discussed earlier, electronic speed controllers (ESCs) are the multi-rotor components necessary to control the speed of a three-phase electric motor. Essentially, after computing the control loop solution, the flight controller has to pass the results to a multi-rotor motor to ensure the UAV is stable and on course. Because all the control-loop dampening and directional predictions are already included in the control loop algorithm, the flexibility of the multi-rotor frame and the motor reaction to the control signals has to be minimized.

The flight controller is connected to an ESC by three wires: ground, power, and controlling signal. The ESCs are controlled by means of pulse width modulation over a signal wire. The ESCs were originally used in RC helicopters in which the motor would be connected to a rotor through a number of gears. To protect these gears from becoming damaged during rapid speed changes, the ESC would compensate and ramp up to the desired speed over some period of time. Also, many ESCs have in-built under-voltage and overheating protection, which switches them off once these extreme conditions are reached. Such ESC behaviour is undesirable for a multi-rotor with a direct rotor drive; the ESC behaviour has to be reliable, regardless of extreme conditions, and simple in operation. Many of the modern ESCs contain an 8-bit microcontroller and can be programmed to react to the controller signal as necessary.

Recognizing these issues with common ESCs, Simon Kirby developed firmware for *ATmega*-based ESCs with superior characteristics for multi-rotor designs. Among the benefits of the ESC, it improves reliability, decreases response time, and increases resolution up to 16-bit output power width modulation with full clock rate. Recent versions of *SimonK* firmware contain a bootloader and can be updated infield through a servo cable. More *SimonK* ESC features can be found on *Github* [41].

MavLink

There are a number of protocols that were designed to handle communication of the flight controller with a ground station. Such protocols are necessary to provide various telemetry and parameter information to and from a ground control station. One of the popular protocols is MavLink, first released in 2009 under the Lesser General Public licence by Lorenz Meier from the Department of Computer Vision and Geometry ETH Zurich, the Swiss Federal Institute of Technology. The protocol is bidirectional and, apart from UAV telemetry such as orientation and position, also carries commands and responses from the drone, essentially allowing total control over the UAV including waypoint navigation. MavLink was designed as a lightweight protocol for serializing C structures and sending them over the serial wire. Subsets of MavLink are well defined and quite extensive.

From a security perspective, what we are interested in is the MavLink Mission Interface, a data format for storing missions to be carried out by an aerial vehicle. The values of the mission interface can be transmitted as waypoints using the MavLink Waypoints Protocol, as well as by individual

actions using a MavLink command message. Among the most interesting of these are MAV_CMD messages allowing execution by the aerial vehicle of commands sent from the ground control station to the UAV. The commands might include MAV_CMD_COMPONENT_ARM_DISARM, which arms or disarms vehicle components such as motors, or MAV_CMD_NAV_WAYPOINT, which is a navigation command directing the UAV to latitude, longitude and altitude (specified as the command parameters). For further detailed analysis, the MavLink specification can be found on the *Pixhawk* site [42].

MavLink has evolved to become very nearly an industry standard among open hardware and software UAV projects. It is currently supported by *ArduPilotMega*, *Pixhawk*, *pxIMU*, *SmartAP* and more [43]. Some of the software packages supporting MavLink include *iDroneCtrl* (*iOS*), *QGroundControl* (*Windows*/*Mac*/*Linux*), *HK Ground Control Station* (*Windows*), *APM Planner* (*Windows*/*Mac*) and *Copter GCS* (*Android*) [44]. Most interesting are MavLink Python bindings, which allows scripting and sending MavLink messages using Python as well as MAVProxy (a plug-in-extendable command-line UAV ground control station). MAVProxy, which is written entirely in Python, is lightweight and allows networking and connecting over multiple computers. It has many useful plug-ins, such as a minimalistic GUI, moving maps, joysticks and antennae trackers [45].

UAVTalk

UAVTalk is yet another lightweight UAV communication protocol [46]. The protocol was originally designed to facilitate UAV communications in the *OpenPilot* project. The protocol is not as actively supported and extensible as MavLink, and at the moment is only used in a small number of UAV projects, namely the *CC3D* [47] and *Revolution* [48] platforms by *OpenPilot* and *Quanton*; and the *FlyingF3*, *FlyingF4* and *Sparky* by *TauLabs* [49]. There are other *STM32F1-*, *F3-*, and *F4*-based platforms that could be ported to run *TauLabs* firmware based on *OpenPilot*. Currently, UAVTalk is bound to the *OpenPilot* ground control station and is not as popular as MavLink.

MultiWii serial protocol

A telemetry and command protocol designed as an alternative to MavLink and for use with *MultiWii* flight controllers. The original goal was to design an even lighter and more compact message exchange protocol than MavLink, one that would be easier to implement in 8-bit embedded systems. The aim of the protocol was to abstract flight controller components from each other, to make it easier to add new modules and to design software for handling communication data.

The protocol messages are binary and header-specific, which essentially should allow for the mixing of various header-specific protocols on the same wire. There's a checksum for each message to make sure the message is not corrupted. The protocol allows the issuing of commands to the flight controller; for instance, MSP_SET_RAW_RC, which essentially allows one to control the UAV over a serial link instead of a radio transmitter. Other interesting commands are: MSP_SET_RAW_GPS, which allows the injection of GPS data into the *MultiWii* control loop; the MSP_SET_PID, which sets PID parameters to defined in the arguments values, and MSP_EEPROM_WRITE, which writes current parameters to EEPROM. The protocol is not currently as popular as MavLink and is predominantly used with *MultiWii* series flight controllers and the accompanying ground control configuration software. The full specification of the MultiWii serial protocol can be found on the *MultiWii* site [50].

As the reader can see from this overview, the rise of UAVs can be credited in part to a high level of interoperability among available componentry, firmware, software and architectures. However, the situation presents significant opportunities for malicious parties to cause a great deal of disruption without requiring them to greatly customize individual attacks. In the next section, we'll examine potential attack surfaces; this paper avoids going into a level of detail that might enable malicious actors to benefit from this work.

ATTACK SURFACES

Bootloaders

After examining a number of flight controller configurations, the possible surfaces of malicious attacks become more apparent. Many flight controllers expose a well-documented bootloader that is not locked to signed firmware. As an example, flight controllers based on *Arduino 2560* implement a subset of the stk500v2 [51] bootloader protocol. The firmware is uploaded over a serial connection to a microcontroller. Systems based on the *STM32F* series, such as *PX4*, use a *PX4* bootloader [52]; meanwhile, *OpenPilot* and *TauLabs* projects rely on a custom bootloader, which can be pushed to the board in rescue mode using the factory ROM bootloader [53]. The *OpenPilot* custom bootloader is responsible for loading firmware, for USB-to-ground-station communications, and for initial set-up of the flight controller hardware [54]. The protocols of the bootloaders are either defined by the chip manufacturers or documented by the developer community. Also, the protocols can be reversed-engineered by looking at the serial communication using any of the serial port sniffers, such as *PortMon* by *Sysinternals* [55] or *Serial Port Monitor* [56]. For protocol analysis one could also look at bootloader source files available in open-source projects.

Firmware

In many cases, firmware can be modified and uploaded to a flight controller to alter its behaviour. There are number of areas in which modified firmware can be made to act maliciously. In the absence of firmware sources, such as in the case of closed-source or hardware systems, the most noticeable impact would be seen when altering the sensor block data stream. Much flight controller hardware tends to use industry-proven and well-documented IMUs such as *InvenSense*'s *MPU6000*/*6050*, *MEAS*'s *MS5611* or *Bosch*'s *BMP085* barometric sensors (mentioned previously), and *Honeywell*'s three-axis *HMC5883L* magnetometer. Many of these sensors are connected using I2C or SPI protocols.

In order to pinpoint a location in firmware where the sensor block communication occurs we could use SPI or I2C sniffers and protocol analysers to record the sensor's byte stream. There are a number of inexpensive solutions for doing so; for instance, the '*Bus Pirate*' from *Dangerous Prototypes* [57] or a simple implementation based on an *Atmel ATTiny2313-*

20PU chip [58]. The author personally likes the solution built with the use of the *Discovery F4* board from *STMicroelectronics*. It is open-source, it can handle communications up to 20MHz, and it communicates using the SUMP protocol [59], which allows the use of many open-source software clients.

There are normally a limited number of I2C and SPI ports available on the core CPU of a flight controller. The sensor block connection can be traced to one of the available CPU ports. Once the byte streams and the communication CPU ports of the sensor block are established, the code serving these ports with the specified byte sequences can be searched for inside the firmware disassembly. Of course, it makes it much easier if the source code of the firmware is available.

Hardware testbed

Sometimes the sensor block or its connections are not easily accessible in the hardware, which makes it difficult to attach a logic analyser probe to the sensors. There are cases where specified flight controller hardware cannot be sourced or is tamper-resistant. In many instances, the specialized hardware can be substituted with readily available development boards. For instance, many *Arduino*-based flight controllers and their firmware can be simulated on a standard *Arduino* board. A number of STM32-based flight controller projects such as *FlyingF3*, *FlyingF4* and *PX4* allow installation of their firmware on *Discovery* series development boards by *STMicroelectronics* [60–62]. Popular sensor chips such as *InvenSense*'s *MPU6050* and *MPU9150*, *MEAS*'s *MS5611*, *Bosch*'s *BMP085* and *Honeywell*'s *HMC5883L* are also available on breakout boards, as shown in Figure 9, providing a convenient way to connect the sensors to development boards and conduct tests and experiments.

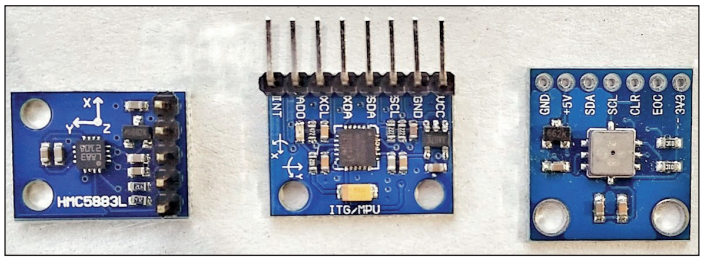


Figure 9: Breakout boards for sensor integrated circuits.

By connecting a breakout sensor board to any of the developer boards mentioned above loaded with the specified flight controller firmware, we could observe the communication protocol using a protocol analyser such as the *Saleae Logic* series [63] or its cheaper available alternatives mentioned above. Having such a configuration makes it much easier to connect to the sensor pins than it would be to do so with the original flight controller boards.

GPS unit

The potential of the GPS unit as an attack surface on UAVs deserves close attention. GPS is one of the most important components of the flight controller system, providing navigational data and stability of the vehicle's position in various flight modes.

There are number of ways in which the GPS operation can be affected or altered. One of the methods is modifying the data inside the firmware programmatically. The GPS is normally connected to the CPU through the UART interface. Most GPS units are well documented and comply with standard protocols of communication, such as NMEA [64] or a binary protocol such as the previously discussed UBX for *u-blox* devices [65]. In some cases, the GPS module contains a network processor that wraps the GPS data in an obscure proprietary protocol. This allows a vendor to maintain hardware exclusivity and charge higher prices for otherwise commonly available modules. In this case, the same methodology applies as it would with any other closed-end sensors. The data can be studied through the serial interface analyser and compared to a similar GPS unit without the additional protocol processor [66].

The other method of altering GPS data involves broadcasting on the frequency of the satellites simulating the GPS signal. This technique either alters the data received by the GPS unit or jams signal reception, making it entirely inaccessible. This method is technically challenging but one of the most effective. The attacker doesn't have to infiltrate a flight controller, the attack can be carried out from a substantial distance and is universal regardless of the flight controller type or the GPS unit. Despite the technical difficulties inherent to the approach, a number of proof-of-concept scenarios have already been demonstrated. These involve the use of software-defined radios and specialized software to simulate the Global Navigational Satellite System (GNSS) signal [67–69].

Telemetry and command feed

The flight controller can be connected to a ground control station over a telemetry and command control link. In many cases, UAV telemetry and command protocol implementations are not inherently secure. The protocols allow UAVs to be reconfigured and controlled remotely using any third-party software, without any special authentication.

There are a few ways an attacker can tap into a telemetry link. One of the methods is to capture, modify and inject back a data stream into a telemetry link connection over a serial port. This data is processed by a ground control station client. Another method is to completely take control of the interface while spoofing the connection for the GCS. All these attacks are carried out at a client site and require access to the ground control station computer. The telemetry feed can be transmitted over the air using any of the available technologies such as Bluetooth, ZigBee, Wi-Fi, or a proprietary radio link.

Two transmission methods that are currently very popular are Bluetooth, for short range communications up to 30 metres, and a radio module, developed by *3DRobotics* [70] and based on the *SiLabs Si1000* chip working on one of the industrial scientific and medical (ISM) bands, for links up to a kilometre. The Bluetooth modules used with the flight controllers are often left configured with their default pairing code. Both of the radio links can be tapped into using a software-defined radio (SDR), such as *HackRF* [71].

Middleware and background processes

With the advent of more powerful and cheaper CPUs, there are a number of flight controller projects that have started to rely on RTOSes such as *NuttX* or even *Linux* as middleware.

This opens the possibility for applications running in the background to essentially gain access to all the sensor drivers, on a par with a flight control application.

Other tactics

Some of the MPUs offered by *InvenSense*, such as the *MPU6000/6050/6500/9150/9255* series, have a unique hardware feature called the Digital Motion Processor (DPM). This is essentially a processing module inside the sensor that can be programmed to perform computations on the data from sensor readings. The DPM firmware is stored in the volatile memory of the MPU and needs to be uploaded to the DPM module on every power-up. This firmware can be modified to alter the chip readings, affecting the flight controller performance. Such modifications are difficult to detect once the firmware is uploaded to the DPM module.

As discussed earlier, *SimonK* ESC firmware can be upgraded over a PWM servo cable. Modifying *SimonK* firmware to, for instance, alter functionality of the ESC once it receives a predefined PWM control signal, could have disastrous consequences.

Because most of the modern controllers are connected over the USB interface, implementing some sort of USB device within its firmware, such a device is susceptible to a BADUSB [72] attack and could carry it out on a client system once the controller is plugged in.

CONCLUSION

In the last few years, we have experienced a shift toward more powerful hardware platforms in embedded designs, and UAVs are no exception. Fuelled by media popularity and the recent advances in sensors, microcontrollers and lithium-polymer batteries, multi-rotors have become more affordable and easier to control. We are witnessing an increase in drone research and development across various types of industries, including agriculture, entertainment, law enforcement and delivery services. On the technical side, the flight controllers, while enjoying healthy competition, are becoming more advanced and powerful. The controllers have begun to rely on popular operating systems as middleware. These include various RTOSs as well as *Linux* and *Android*.

Legislation around UAV use is wildly uneven around the world, with researchers in some countries feeling compelled to do their testing in nations that are currently less restrictive about the public use of UAVs. In the US, faced with the inevitability of UAV omnipresence, the Federal Aviation Administration is the federal entity that will regulate the commercial and amateur drone industry. To that end, early in 2015 the agency issued its long-awaited 'Notice 8900.291 – Inspection and Maintenance Program Requirements for Airworthiness Certification of Unmanned Aircraft Systems Operating Under 55 Pounds' [73]. In the two-page notice, released in March 2015, the FAA states its intent to take an incremental approach to gaining 'a better understanding of operational issues such as training requirements, operational specifications, airworthiness, and technology' where UAVs are concerned, while pledging to work toward integrating UAVs into the nation's airspace. The proposed regulations themselves cover such matters as commercial operator licensing, top altitude and speed, and flight paths [74]. While this appears to point to an eventual relaxing of the US's current airspace restrictions on UAVs, it is by no means clear that 'technology' in this context includes potential security concerns.

With the increase in availability of such devices and their presence in various areas of our life comes a responsibility to harden and secure these platforms from malicious attacks and rogue software. Special consideration must be given to securing firmware on embedded UAV modules. Best practices must be followed in securing such firmware, starting from its architectural design, implementation, and software development processes. The use of secure boot loaders and mechanisms of firmware authentication and encryption must become ubiquitous. Attention must be paid to the uses of encryption for wireless control and telemetry protocols. Above all, we have to realize that paying the cost for securing firmware and embedded devices upfront can prove much cheaper than trying to mitigate a disaster resulting from inadequate security measures – especially in the case of UAV.

REFERENCES

[1] http://www.tealgroup.com/index.php/about-teal-group-corporation/press-releases/118-2014-uav-press-release.

[2] http://www.extremetech.com/extreme/171879-amazon-unveils-30-minute-prime-air-quadcopter-delivery-service-but-its-completely-impractical.

[3] http://readwrite.com/2015/03/17/google-x-astro-teller-sxsw-drone.

[4] http://www.wired.co.uk/magazine/archive/2015/03/start/where-the-drones-roam.

[5] http://en.wikipedia.org/wiki/PID_controller.

[6] http://www.kkmulticopter.kr/index.html?modea=credits_kk.

[7] http://www.rcgroups.com/forums/showthread.php?t=1675613.

[8] http://www.atmel.com/images/doc0943.pdf.

[9] http://www.multiwii.com/wiki/index.php?title=Hardware.

[10] https://code.google.com/p/multiwii/.

[11] http://www.multiwii.com/wiki/index.php?title=Mods#STM32_32-bit.

[12] https://github.com/cleanflight/cleanflight.

[13] https://code.google.com/p/multiwii/source/browse/#svn%2Ftags.

[14] https://code.google.com/p/mw-wingui/.

[15] https://chrome.google.com/webstore/detail/baseflight-configurator/mppkgnedeapfejgfimkdoninnofofigk?hl=en.

[16] https://chrome.google.com/webstore/detail/cleanflight-configurator/enacoimjcgeinfnnnpajinjgmkahmfgb.

[17] http://www.multiwii.com/forum/viewtopic.php?f=8&t=1516.

[18] https://www.openpilot.org/.

[19] https://www.openpilot.org/products/openpilot-Revolution-platform/.

[20] http://taulabs.org/.

[21] http://diydrones.com/.

[22] http://makezine.com/2014/10/13/dronecode-linux-corporation-3d-robotics-create-open-source-uav-software-endeavor/.

[23] http://www.u-blox.com/images/downloads/Product_Docs/u-blox6_ReceiverDescriptionProtocolSpec_(GPS.G6-SW-10018).pdf.

[24] https://tools.ietf.org/html/rfc1145.

[25] http://copter.ardupilot.com/wiki/common-autopilots/common-apm25-and-26-overview/.

[26] https://pixhawk.org/modules/pixhawk.

[27] http://www.st.com/web/catalog/sense_power/FM89/SC1288/PF254039.

[28] http://nuttx.org/.

[29] http://hackaday.com/2014/06/06/droning-on-flight-controller-round-up/#comment-1555458.

[30] http://firmware.diydrones.com/Tools/APMPlanner/.

[31] http://firmware.diydrones.com/Tools/MissionPlanner/.

[32] http://firmware.diydrones.com/.

[33] http://www.emlid.com/.

[34] http://en.wikipedia.org/wiki/DJI_%28company%29.

[35] http://www.dji.com/products.

[36] http://www.dji.com/info/spotlight/whats-difference-of-naza-m-litenaza-m-v1naza-m-v2.

[37] http://www.rcgroups.com/forums/showthread.php?t=2071772.

[38] http://www.rcgroups.com/forums/showthread.php?t=1995704.

[39] http://www.rcgroups.com/forums/showthread.php?t=2331009.

[40] http://naza-upgrade.com/.

[41] https://github.com/sim-/tgy.

[42] https://pixhawk.ethz.ch/mavlink/.

[43] http://www.qgroundcontrol.org/mavlink/start.

[44] http://www.qgroundcontrol.org/mavlink/start.

[45] http://tridge.github.io/MAVProxy/.

[46] https://wiki.openpilot.org/display/WIKI/UAVTalk.

[47] http://www.openpilot.org/products/openpilot-coptercontrol-platform/.

[48] http://www.openpilot.org/products/openpilot-Revolution-platform/.

[49] https://github.com/TauLabs/TauLabs/wiki.

[50] http://www.multiwii.com/wiki/index.php?title=Multiwii_Serial_Protocol.

[51] http://www.atmel.com/images/doc2591.pdf.

[52] https://pixhawk.org/dev/px4_bootloader.

[53] http://www.st.com/web/en/resource/technical/document/application_note/CD00167594.pdf.

[54] https://wiki.openpilot.org/display/WIKI/Bootloader+Update.

[55] https://technet.microsoft.com/en-us/sysinternals/bb896644.aspx.

[56] http://www.serial-port-monitor.com/.

[57] http://dangerousprototypes.com/bus-pirate-manual/.

[58] http://en.radzio.dxp.pl/i2c-sniffer/.

[59] http://www.sump.org/projects/analyser/protocol/.

[60] https://github.com/TauLabs/TauLabs/wiki/Creating-a-FlyingF3-from-scratch.

[61] https://github.com/TauLabs/TauLabs/wiki/Creating-a-FlyingF4-from-scratch.

[62] https://pixhawk.org/modules/stm32f4discovery.

[63] https://www.saleae.com/logic/.

[64] http://www.gpsinformation.org/dale/nmea.htm.

[65] https://www.u-blox.com/images/downloads/Product_Docs/u-bloxM8_ReceiverDescriptionProtocolSpec_%28UBX-13003221%29_Public.pdf.

[66] http://www.rcgroups.com/forums/showthread.php?t=1995704.

[67] http://www.navsys.com/Papers/13-09-001_Open_Source_SDR_Platform_for_GNSS_Recording_and_Simulation.pdf.

[68] http://gpsworld.com/defensesecurity-surveillanceassessing-spoofing-threat-3171/.

[69] http://gpsworld.com/drone-hack/.

[70] https://store.3drobotics.com/products/3dr-radio-915_MHz.

[71] https://greatscottgadgets.com/hackrf/.

[72] https://srlabs.de/badusb/.

[73] https://www.faa.gov/regulations_policies/orders_notices/index.cfm/go/document.information/documentID/1027090.

[74] FAA Releases Proposed Drone Regulations. Drone 360 (magazine), Special Issue 2015 (Vol. 1 Issue 1).