IOActive®

Research-fueled Security Services

\ RESEARCH \

Drone Security and Fault Injection Attacks

Gabriel Gonzalez
Director of Hardware Security

June 2023

Table of Contents

SUMMARY **3**

INTRODUCTION **4**

ATTACK SURFACE **5**

OcuSync 5
Mobile Apps 6
Assets 6
Drone Architecture 6

TECHNICAL BACKGROUND **8**

XYZ Table 9
EM Probe 9
Oscilloscope 11
Spider and EM-FI Probe 12
Target Drone 12
Drone PCB Preparation 13

FIRST APPROACH **14**

SECOND APPROACH **18**

FUTURE WORK **22**

MITIGATIONS **23**

APPENDIX A: ADDITIONAL INFORMATION **24**

About IOActive 24
About the Author 24

APPENDIX B: JAVA CODE **25**

APPENDIX C: FIPY CONTROLLER CODE **29**

©2023 IOActive, Inc. All Rights Reserved.

Abstract

The use of Unmanned Aerial Vehicles (UAVs), commonly referred to as drones, continues to grow. Drones implement varying levels of security, with more advanced modules being resistant to typical embedded device attacks. IOActive's interest is in developing one or more viable Fault Injection attacks against hardened UAVs.

This paper covers IOActive's work in setting up a platform for launching side-channel and fault injection attacks using a commercially available UAV. We describe how we developed a threat model, selected a preliminary target, and prepared the components for attack, as well as discussing what we hoped to achieve and the final result of the project.

Summary

IOActive set out to explore the possibility of achieving code execution on a commercially available drone with publicly disclosed vulnerabilities using non-invasive techniques, such as electromagnetic (EM) side-channel attacks and EM fault injection (EMFI). If successful, we could apply the lessons learned to a completely black-box approach and attempt to compromise devices with no well-known vulnerabilities.

As a target, we chose DJI, a seasoned manufacturer that emphasizes security in their products,[1] such as signed and encrypted firmware, Trusted Execution Environment (TEE), and Secure Boot. We used a controlled environment to investigate the impact of side-channel attacks and EMFI techniques.

We demonstrated that is feasible to compromise the targeted device by injecting a specific EM glitch at the right time during a firmware update. This would allow an attacker to gain code execution on the main processor, gaining access to the Android OS that implements the core functionality of the drone.

[1] https://security.dji.com/data/resources/

©2023 IOActive, Inc. All Rights Reserved.

Introduction

Drones, also known as Unmanned Aerial Vehicles (UAVs), are increasingly being deployed in industries like aviation, agriculture, and law enforcement. As the use of these devices has expanded, so have cybersecurity concerns. These versatile machines offer many benefits, but they also present unique challenges when it comes to maintaining security.

One of the key challenges is the fact that drones are often operated remotely. This means there is a potential for external actors to gain access to the drone's control systems by intercepting the wireless signals used to control the drone, compromising the ground-based computer systems used to operate the drone, or gaining access to the device itself; if a drone is stolen, the thief could gather sensitive information or plant malware on the system.

This whitepaper covers IOActive's research into the current security posture of the drone industry. For this work, we chose one of the most common drone models, DJI's Mavic Pro.[2]

The goals of this research were to:

- Learn about the architecture of UAVs
- Investigate UAVs' weakest points
- Determine what approaches make sense in real-world attacks
- Gain experience launching EMFI attacks against high-end, multicore systems running full-fledged, multi-task operating systems, vs microcontrollers running bare metal software

This whitepaper covers the following areas:

- Attack Surface: Provides an overview of the drone's components and identifies valuable assets
- Technical Background: Introduces concepts about the targeted drone and test setup
- First Approach: Describes IOActive's attempts to use side-channel attacks to acquire the firmware decryption keys
- Second Approach: Describes IOActive's attempts to use EMFI to achieve code execution
- Future Work: Discusses next steps in IOActive's research
- Mitigations: Recommendations to address the highlighted issue

.

[2] https://www.dji.com/mavic

©2023 IOActive, Inc. All Rights Reserved.

Attack Surface

Drones are used in variety of applications, including military, commercial, and recreational. Like any other technology, drones are vulnerable to various types of attacks that can compromise their functionality and safety.

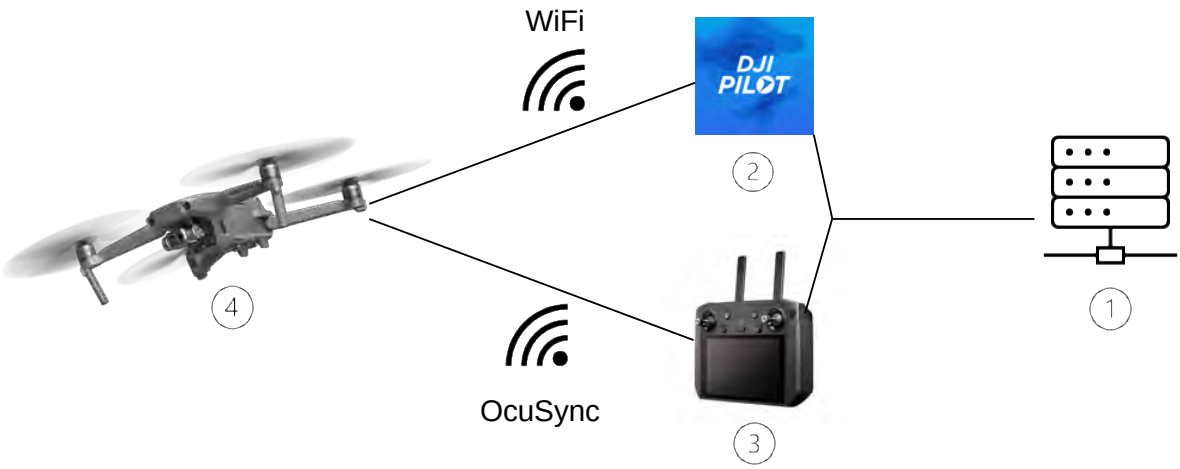


Figure 1: High-level overview of the elements involved in the normal operation of a drone

As seen in Figure 1, drones expose several attack surfaces, depending on their specific capabilities. Some common attack surfaces include:

1) **Backend**: Like almost any other modern system, drones are vulnerable to a variety of attacks against their backend systems. Attackers can exploit a range of issues (such as SQL injection or server-side request forgery) to gain access to sensitive data, manipulate the system, or disrupt operations. Additionally, attackers may be able to use the backend system as a pivot point to launch further attacks on other components of the ecosystem.

2) **Mobile Apps**: Attackers can exploit vulnerabilities in mobile apps to gain access to sensitive data or remotely control the drone. Common ways to break into these would be vulnerabilities in the applications or operating system, and even targeted attacks.

3) **Radio Frequency (RF) Communication**: RF communications systems are used by drones to receive and transmit commands, data, and video. RF attacks can use various techniques, such as jamming, spoofing, and interference, to disrupt or manipulate the RF signals used by drones. DJI implements a custom RF protocol, OcuSync, to send video and commands to and from the remote controller used to control the drone.

4) **Physical Device**: Hardware or software vulnerabilities could allow an attacker to access key aspects of the drone, such as the firmware and sensitive information, and potentially modify its behavior.

OcuSync

OcuSync is a wireless communication protocol developed by DJI. It is used in many of DJI's drones, including the popular Mavic series, to provide a reliable and low-latency link between the drone and its associated remote controller.

One of the key features of the OcuSync protocol is its ability to support multiple communication channels. This allows the drone to automatically switch between channels to maintain a strong and stable connection, even in challenging environments where radio interference may be present.

©2023 IOActive, Inc. All Rights Reserved.

The OcuSync protocol also includes security features to protect over-the-air (OTA) communications. This includes the use of encryption to protect the wireless link between the drone and the remote controller, as well as authentication mechanisms to prevent unauthorized users from gaining access to the drone's control systems.

Mobile Apps

Mobile apps are available for both iOS and Android devices, and they allow users to easily control and monitor their drones from a smartphone or tablet.

Some of the key features of DJI's mobile apps are the ability to remotely control the drone's movement, access camera settings and control the camera, and view a live video feed from the drone's camera.

In addition to these basic control features, DJI's mobile apps also offer a range of advanced capabilities, such as automatic flight planning, custom flight path creation, and the ability to access and manage the drone's flight logs and other data.

Assets

IOActive's initial goal was to gain access to the drone's firmware and search for vulnerabilities in the exposed attack surface. The following are assets that an attacker may target, depending on the desired outcome.

Firmware

The software running on the System-on-Chip (SoC) in the target device is the holy grail for vulnerability research. By reverse engineering the most recent version of the software that manage communications, it is possible to gain a greater understanding of the system and identify potential security flaws and vulnerabilities.

For the Mavic Pro, DJI only provides signed and encrypted firmware packages. A publicly available exploit for the Mavic Pro[3] permitted IOActive to enable debugging access to the device. This information allowed us to better understand the internals of the device and move forward with our research.

No-fly Zones

Most UAVs implement several "no-fly zones" to help prevent accidents and unauthorized use. These no-fly zones are areas where the drone is not allowed to take off or fly, and they are designed to protect sensitive locations such as airports, military bases, and other areas where drone operation may pose a safety risk.

Sensitive Information

The utilization of drones to store information, such as flight plans, images, and other potentially sensitive data, warrants consideration in certain contexts.

Drone Architecture

DJI incorporates a range of sensors and peripherals in its drones to enable advanced capabilities and improve performance.

[3] https://github.com/CunningLogic/DUMLRacer

©2023 IOActive, Inc. All Rights Reserved.

- **Gyroscopes and accelerometers** are used to measure the drone's orientation and movement, providing the information needed to maintain stability and control.
- **Barometers** measure atmospheric pressure, allowing the drone to determine its altitude and maintain a consistent flight height.
- **Global Positioning System (GPS)** sensors are used to determine the drone's location and enable features such as automatic takeoff and landing and geofencing.
- **Compass** sensors are used to determine the drone's heading and enable features such as automatic waypoint navigation.
- **Ultrasonic** sensors are used to measure the distance to the ground or other objects and enable features such as automatic hovering and obstacle avoidance.
- **Cameras** are used to capture photographs or video footage from a bird's-eye view. Drones are equipped with cameras to allow users to see what the drone sees in real time and to capture images and video from unique perspectives that would not be possible with a traditional camera

The two main processors in DJI's Mavic Pro are:

1) **Video and Image Processing CPU**: This SoC is produced by Ambarella International LP (Ambarella), a technology company that specializes in the design and development of low-power, high-definition video processing semiconductors.[4] The company was founded in 2004 and is headquartered in Santa Clara, CA. Ambarella's products are used in a range of applications, including security cameras, drones, and sports cameras. The company's chips are known for their ability to capture and transmit high-quality video while consuming very little power.
2) **Android-based Control CPU**: This is an ARM Cortex-A7-based SoC by Leadcore Technology Co. Ltd. (Leadcore), a Chinese telecommunications company that is based in Beijing. The company is known for its work in developing telecommunications technology, including mobile phone chipsets and other related products. Leadcore has been in operation since 2009, and its products include a line of mobile phone chipsets used in a variety of smartphones and other devices.

[4] https://www.ambarella.com/

©2023 IOActive, Inc. All Rights Reserved.

Technical Background

Side-channel attacks rely on indirectly obtaining information about a target system by taking different types of measurements during the execution of specific operations. Some common attacks that use side-channel analysis are:

- **Timing Attacks**: The time a targeted operation takes to complete is analyzed and exploited, for example, to guess a Personal Identification Number (PIN) or break cryptographic implementations.
- **Power Analysis**: Simple Power Analysis (SPA) and Differential Power Analysis (DPA) exploit the power consumed by targeted operations. Measurements are usually taken by tapping the voltage path of the chip. The captured data is mathematically processed to recover secrets, usually cryptographic keys.
- **EM Analysis**: Instead of retrieving power consumption directly from the voltage rails of the hardware, an EM probe is placed close enough to the chip to retrieve EM emanations. This method has the advantage of being less invasive and more localized than power analysis.

Electromagnetic Fault Injection (EMFI) aims to cause a disruption in the hardware when processing certain operations. A metal coil (EM probe) is placed close to the surface of the target processor. The current flow through this coil will induce current changes inside the processor. This is expected to elicit changes in the behavior of the CPU in a way that could be used to gain an advantage; for example, by changing a value in a memory cell or register before or after it is processed.

As mentioned above, the advantages of EMFI include its non-invasive nature and that the perturbation is more localized, as opposed to power glitching, where the PCB typically requires physical modifications and the glitch affects all internal peripherals and processing units connected to the targeted power rail.

For this work, IOActive used the widely recognized FI/Side-Channel Analysis (SCA) suite from Riscure.[5] In particular, we used the components described in the remainder of this section, which are accompanied by a powerful and comprehensive software suite designed to support security researchers.

[5] https://www.riscure.com/security-tools/

©2023 IOActive, Inc. All Rights Reserved.

XYZ Table

When using power analysis techniques, the data is retrieved by tapping the voltage path the target processor uses to power all the internal processing units. One of the challenges when transitioning from voltage to EM readouts is finding the correct position on the surface of the chip. To accomplish this, we utilized an XYZ table, shown in Figure 2, to scan the chip area and locate the optimal spots where more information is leaked.

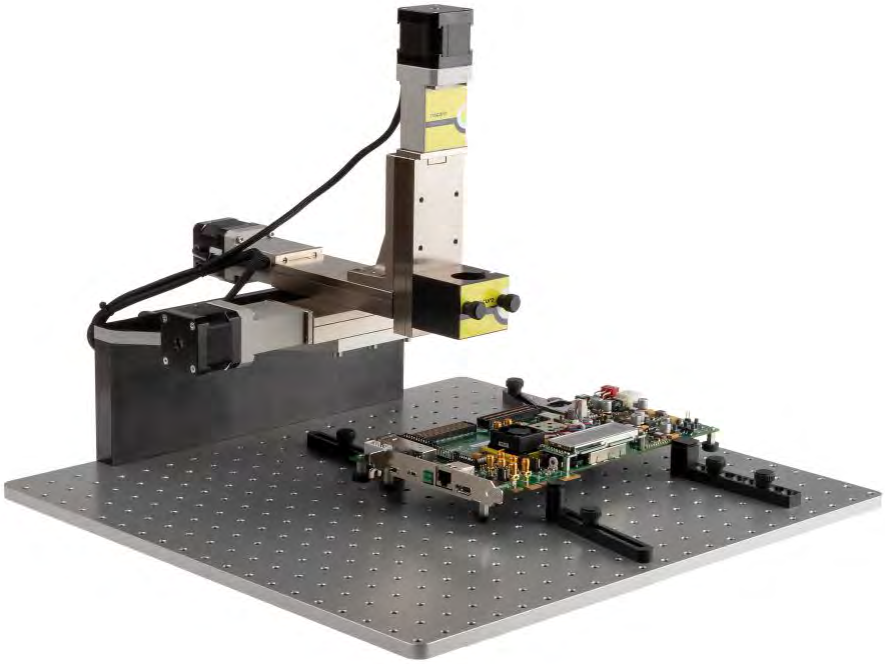


Figure 2: XYZ table with a PCB secured to the bottom surface

In order to correctly use this device, it is essential to properly set the Z-axis. This is a critical factor, as if the EM tip is too close to the target, the tip itself could be damaged while moving around the chip's packaging. On the other hand, if the EM tip is too high, the measured signal will be too weak to be useful.

IOActive employed two techniques to address this issue. The first involved using a USB micro-camera to observe the gap between the EM tip and the surface. The second was a technique commonly used in 3D printing: placing a thin sheet of paper between the EM tip and the chip, positioning the EM tip, and then attempting to pull the paper out. If the gap is too small, the paper will not move; conversely, if it is too large, it will move freely. A small amount of friction when moving the paper indicates an optimal tip height.

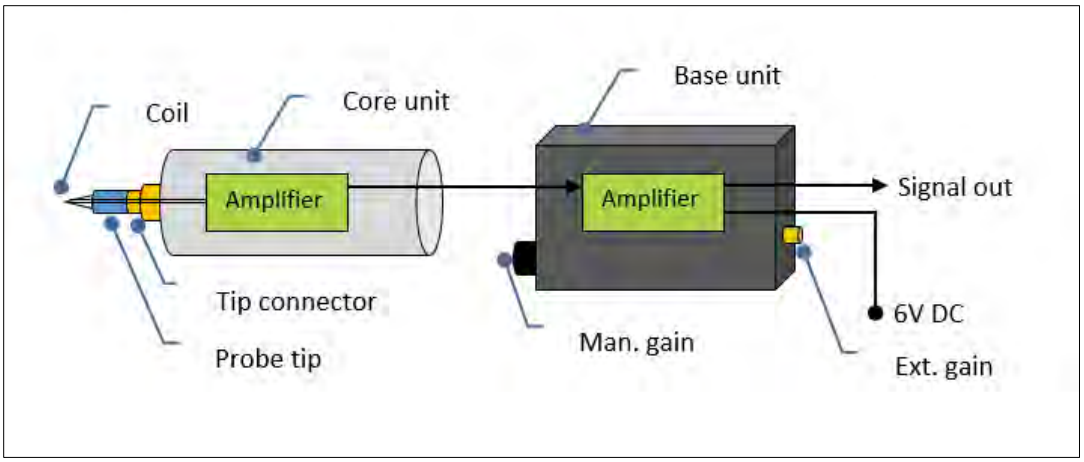
Once the Z-axis has been set to a fixed position, the next step is to specify a rectangle on the XY axis, that is the chip area to be scanned.

EM Probe

The high-precision EM probe IOActive used for this research is designed to detect EM emissions from semiconductor circuits, and is equipped with three tips of varied diameters (0.2mm, 0.5mm, and 1.2mm) that possess a directed coil and a protective Teflon shell. This tool is capable of picking up EM fields of up to 6GHz and converting them into an AC signal.

©2023 IOActive, Inc. All Rights Reserved.

The EM probe is connected to an amplifier, as illustrated in Figure 3, and the output is directed to the oscilloscope via a BNC cable. The software suite then interprets the EM measurements to create a heat map in order to pinpoint the area of maximum emanation.



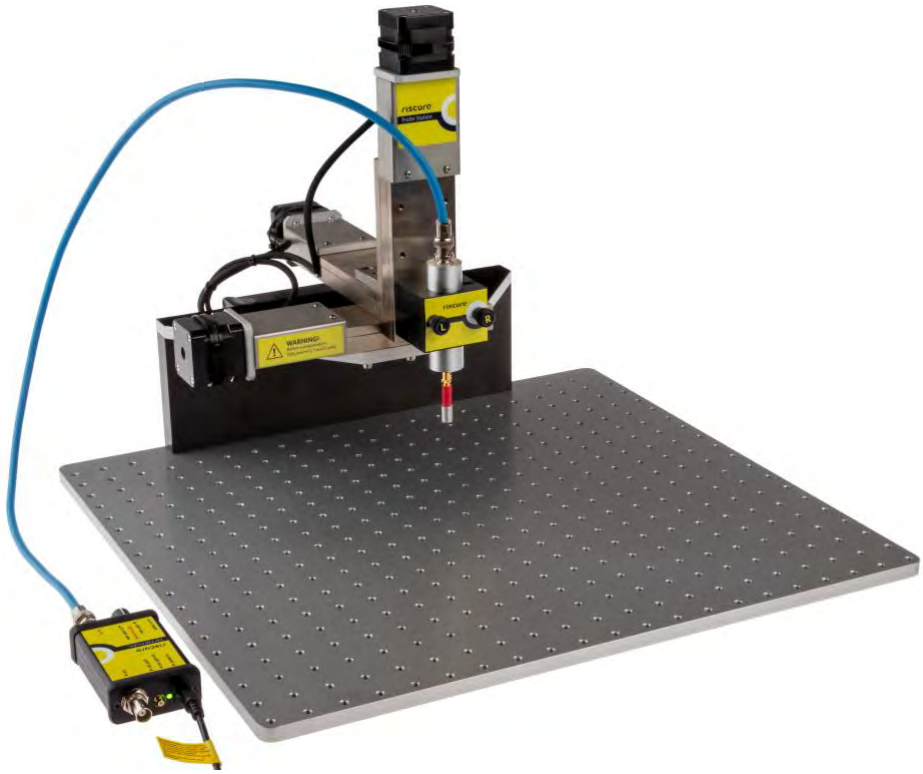


Figure 3: Block diagram of the EM probe and amplifier and the XYZ table with the EM probe connected to the amplifier

©2023 IOActive, Inc. All Rights Reserved.

Oscilloscope

IOActive used the oscilloscope shown in Figure 4 as a key element in both the SCA and FI portions of the project.

Figure 4: Oscilloscope used to identify sweet spots and verify EM glitches

During SCA, the oscilloscope was used to capture signal leakages from the computations performed in the target CPU. During EMFI, it was used to verify that glitches were generated with the desired shape and timing.

©2023 IOActive, Inc. All Rights Reserved.

Spider and EM-FI Probe

For EMFI, IOActive used Riscure's Spider tool and EM-FI Transient Probe, as shown in Figure 5.

Figure 5: Spider (left) and EM-FI Transient Probe (right) from Riscure

Spider reduces complexity in SCA and EMFI by creating a single control point with all the I/O and reset lines for custom or embedded interfaces. As part of this project, IOActive used the Spider tool to generate arbitrary glitch waves. The tool allows for adjusting the output voltage, timing, repetition rate/frequency, and duration of the pulse.

Riscure's EM-FI Transient Probe induces fast, high-power, EM pulses on a user-defined location of the chip. Fast and short pulses are configurable from software, providing a fast and predictable response to trigger.

Target Drone

IOActive's approach was to use a controlled testing environment to investigate whether it is feasible to attack a hardened drone via side-channel or EMFI techniques. In order to ensure a controlled environment, IOActive targeted a drone model and firmware version with publicly known vulnerabilities. This would enable us to gain a better understanding and control each of the different stages by reverse engineering the software components involved in the firmware upgrade process. Furthermore, it would provide an opportunity to gain insight into potential device malfunctions or other behaviors that could delay testing. If successful, we could take the lessons learned in this phase and attempt these types of attacks on devices with no well-known vulnerabilities using a completely black-box approach. In addition, if any future targets use the same SoC as the initial target, we could even reuse the search area identified during this project. This would save valuable time by eliminating the need to scan the whole SoC surface and tune parameters such as duration, delay, and strength.

Given these factors, IOActive selected the DJI Mavic Pro. In addition to having leaked keys, it is widely availability on the second-hand market, which could be beneficial in the event of an accident resulting in damage to the unit during our research process.

©2023 IOActive, Inc. All Rights Reserved.

Drone PCB Preparation

The first step was to identify the drone's main PCB and create an isolated environment where it could be powered without the original battery and placed under the analysis probes.

After removing the plastic cases, peripherals, and sensors, the result is shown in Figure 6. The component we were interested in is outlined in red: the Leadcore SoC. This SoC runs a custom version of Android and handles USB communications and the firmware upgrade process.

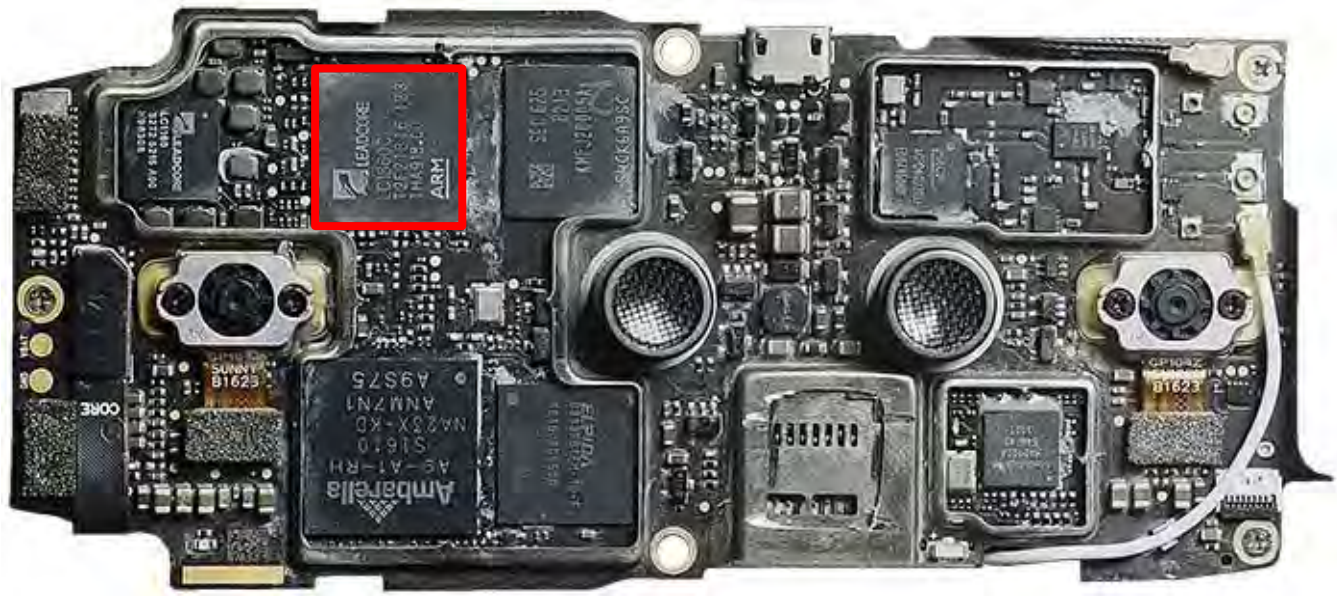


Figure 6: Main PCB with all the different components[6]

[6] Source: https://github.com/o-gs/dji-firmware-tools/wiki/WM220-Core-Board-A

©2023 IOActive, Inc. All Rights Reserved.

Figure 7 shows the PCB under analysis. It should be noted that IOActive had to include an external fan to dissipate the heat generated by the PCB components. Without it, the temperature would quickly rise after power-up, leading to a reboot.

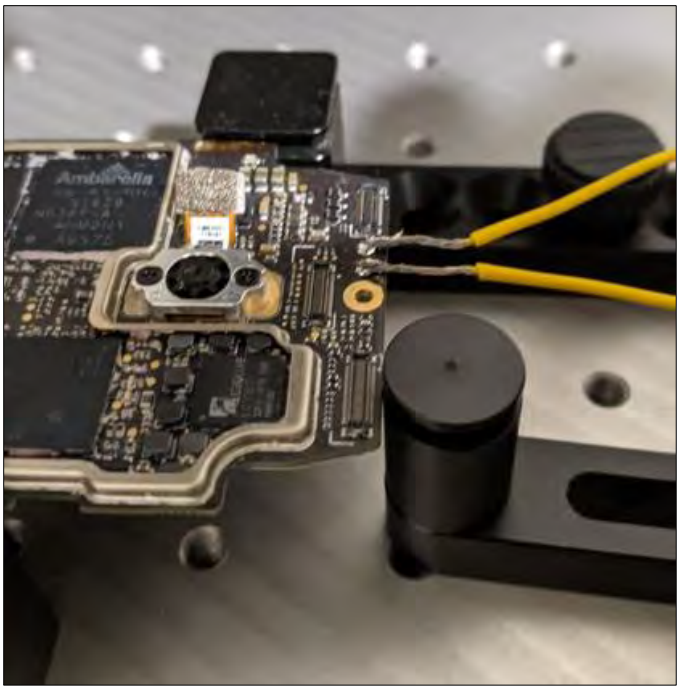


Figure 7: PCB on the XYZ table with power from an external power supply

First Approach

As mentioned in the Assets section, one of the main priorities of any threat actor when approaching a system is gaining access to unencrypted firmware. Therefore, our first goal was to study the feasibility of recovering the keys used to encrypt/decrypt the firmware packages using side-channel power leaks.

Power leakages on cryptographic algorithms are tied to mathematical operations using an attacker-controlled payload and the unknown key. The instructions where data from both elements are computed will consume more or less power depending, for example, on the number of bits in the result. This is typically called the leakage model and is sometimes modeled as the hamming distance.

IOActive's goal was to construct a setup where we could:

1) Send random data to the target to be used later in cryptographic computations
2) Perform repeated tests to collect thousands of power traces
3) Analyze the retrieved data and recover the encryption key

As mentioned step 1, a key part in exploiting side-channel leakage is for the targeted operation to process data with enough entropy to be statistically analyzed. In this case, we had the ability to submit random data that would hit the targeted operations in the cryptographic algorithm. To achieve this, we analyzed the format of the package and generated valid packages that could be fed into the decryption process. Figure 8 shows one of the packages and some of the values that were modified to feed the verification application.

©2023 IOActive, Inc. All Rights Reserved.

```
    00010203 04050607 08090A0B 0C0D0E0F 10111213 14151617 18191A1B 1C1D1E1F 20212223 24252627
000 494D2A48 01000000 00050000 00000000 E0000000 00010000 20030000 00050000 00000000 01000000  IM*H                  .
028 5052414B 5055454B 24151CAE 6D0D9D41 0A571369 A9A94E05 30313030 00000000 00000000 00000000  PRAKPUEK$  .m .A W i..N 0100
050 00000000 00000000 00000000 00000000 00000000 42380602 08121720 00000000 00000000 00000000                      B8
078 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 01000000
0A0 B98F56C3 3E0C4125 638C5238 CE46F429 CF4A4B9D CE158889 4E47061D 22561F70 30313030 00000000  ..V.> A%c.R8.F.).JK.. ..NG  "V p0100
0C8 644EE800 00000000 00000000 00000000 00000000 00000000 7DF37594 B62C1C01 6D31244D E96D268B  dN.                      }.u..,  m1$M.m&.
0F0 A9151448 27961721 D71FE2FB 206EF58B EC945BF4 E448119C 73361526 31817B33 FD39C5CA AB5A742E  .  H'. !. .. n....[..H .s6 &1.{3.9...Zt.
118 FD331CF0 7BE12DA6 74407FCA 3114B0BE A91FEDD2 1E609ACC 00000000 CC7AE80D 4B341831 ED65720A  .3 .{.-.t@ .1 .... .. `..     .z. K4 1.er
140 028568DB 73F7C629 FA49CC12 E7BF3D6A 175165EC CA33CB6D 707C2082 0179701A D8FD4088 A8F5C59C   .h.s..).I. ..=j Qe..3.mp| . yp ..@.....
168 469E4FEB 6D3DD5A4 521D53BA 68156872 D78C5AC6 685FDCE3 6871CD96 CD025C33 D7607C77 8F933D76  F.O.m=..R S.h hr..Z.h_..hq... \3.`|w..=v
190 78557330 7BE7FAA7 69959FE0 3D0DAF43 01BF2AB4 024B6032 3E392504 51D169EB E8B5B892 D73B8587  xUs0{...i...= .C .*. K`2>9% Q.i......;..
1B8 A5463A3C F6E5BB12 AE362F65 57D61035 42976BCE 6C3A7656 A85A3054 72FD5590 649BBB7B 952E8DF2  .F:<... .6/eW. 5B.h.l:vV.Z0Tr.U.d..{....
1E0 18F5B121 5F32481D 80602B31 324AE41C CFC17CD8 1544F254 E3D7F9FB 45668566 BE47A45D 9008E41C   ..!_2H .`+12J. ..|. D.T....Ef.f.G.]. .
208 6DA77A9C 78D27CBF 4426CD12 453483CA 494CB9D2 F632AC54 CCB971BE 9C3E2837 DA4327EB AECD7FD9  m.z.x.|.D&. E4..IL...2.T..q..>(7.C'... .
230 F5000714 4BE458BC 4683131C 43895184 E72BE536 C327F74D BB952A5C 4EF87617 C223793B DE7996D4  .    K.X.F.  C.Q..+.6.'.M..*\N.v .#y;.y..
258 AA76D1AA 60ECC9A9 7EB93B51 2DF0837F 95C65441 AA6FDB11 CCDA404F F84695D4 2E06C4E4 CD895A4A  .v..`...~.;Q-.. ..TA.o. ..@O.F... ....ZJ
280 BBAF6B4A 776EE185 F1134D35 F24A8103 D0F7A28B D06EDA49 46415795 21C46151 FC399484 C336CAB5  ..kJwn... M5.J. .....n.IFAW.!.aQ.9...6..
2A8 3A30AF3C A57E3E9F 7D3C6739 C5376038 52F4C04B DE22890C 068D1B3E FCC90BC9 009C20F3 00613427  :0.<.~>.}<g9.7`8R..K.".  . >.. . .. a4'
2D0 417B8044 912F0254 B0313ED5 B37D9371 A4114025 59D434F4 41418B75 1B86D4B7 985D9D3F A2FBBD91  A{.D./ T.1>..}.q. @%Y.4.AA.u ....].?....
2F8 7D65CAB2 4976AAAD 9CFF0294 7018BCAB 91C1F7ED 1757ACEF 349E4187 3B0628C3 D984DDBD B1528E2B  }e..Iv.... .p ..... W..4.A.; (......R.+
320 630ACB5D 2DA3E53B 4065FBB0 606191C8 EB74C0C8 177F2CF1 6D909FFC EC688F62 929CC697 8FF861F2  c .]-..;@e..`a...t..  ..m....h.b......a.
348 2C5E0F8E 6AD27B43 C9DBF24E B7DDDF88 F294C8E9 98A97A69 7216A638 479C4D5E E135E264 6AD906A2  ,^ .j.{C...N.........zir .8G.M^.5.dj. .
370 AD854F76 4204898B D39BA4CA 61D45E93 DF7DCAD2 E14A3277 D05BE58F B969A2E4 4B016686 18A1C8C0  ..OvB ......a.^..}...J2w.[...i..K f. ...
398 E4D805A1 27DD47D5 4EC79DE7 2511DE2A 7D76EDDC 853E867C 77861660 D0D016F5 278C5323 6EB50566  .. .'.G.N...% .*}v...>.|w. `.. .'.S#n. f
3C0 D6D61128 7910E164 DD82BC70 3D9E6DEE 0852E144 9EB06861 73A04334 0FB24C16 052B7916 3E8770B7  .. (y .d...p=.m. R.D..has.C4 .L  +y >.p.
3E8 1ED797D5 AE07A593 E410FEBE 9743BEC4 B44CC654 64A1F485 31DC3796 250D6011 E488918E 52E2748A   .... ... ...C...L.Td...1.7.% ` ....R.t.
410 AD730165 3E18F5F9 4E0C57B2 CCB87E60 C6E7025D A9358D79 FA0FBB6A C27F8825 EF997D6A 698D46FB  .s e> ..N W...~`.. ].5.y. .j. .%..}ji.F.
438 1C54C115 4E4E5BC3 AEC6CDA3 1E3E11EE 73EC1B11 B0B0AF09 2366A9CA 4E04EADF C9503A48 E4729D9D   T. NN[..... > .s.  ... #f..N ...P:H.r..
460 79C7B018 603A5920 9DEC7C1E 6E70AFDA 48059951 9547F1AA DD0B4EE7 2A610EF0 B7E9DA7E 94F872A0  y.. `:Y ..| np..H .Q.G... N.*a ....~..r.
488 0521491E FB22ECE8 6D035388 5417EA78 D4969D59 9F915610 A4D391B6 3912294C B1EEBA8A 53486DCB   !I ."..m S.T .x...Y..V ....9 )L....SHm.
4B0 18B5834C 2180C07B 2516C804 E9DA8CCA A261DA97 BC63A940 CC8F22DC D0B234F5 1D1F7C95 CC59720B   ..L!..{% . .....a...c.@.."...4.  |..Yr
4D8 8A0F22C9 D4C6397F 06DFE394 616A8DFD 624D552E 56984598 797587DA F6A20A4E 22BD9A88 5A9C9BF4  . "...9  ...aj..bMU.V.E.yu.... N"...Z...
500
```

Figure 8: Binary view of upgrade packages showing the type of package and length

The generated packages contain random data that would be used during statistical analysis to correlate with the captured EM traces. Therefore, we needed to generate a large number of "fake" update packages.

Working on step 2 involved understanding the update process so we could automate the task of submitting modified packages and recording EM emanations for later analysis.

Figure 9 describes the steps involved in the firmware update process on a DJI drone. Since we were lucky enough to get a model with an ADB shell running on it, we could identify the process dealing with the signature and decryption of the software package.

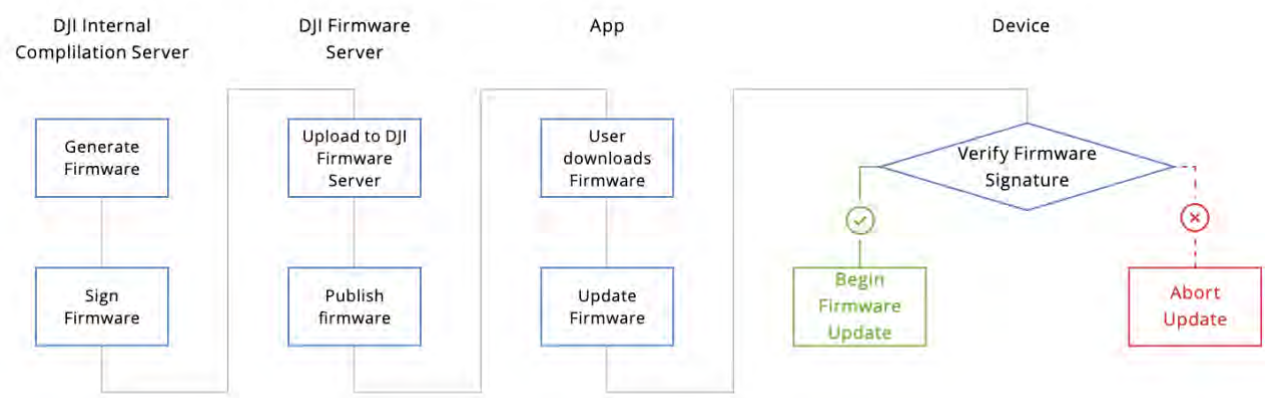


Figure 9: Steps involved in the firmware update process[7]

[7] Source: DJI Security White Paper: https://security.dji.com/data/resources/

©2023 IOActive, Inc. All Rights Reserved.

Signature verification and decryption were performed by the `dji_verify` application. This executable is designed to run as a command-line tool and accept various arguments, including the path to the file containing the firmware package.

Further analysis of the binary allowed us to identify the point where signature verification was performed. Since one task is to find the spot on the chip's surface where the EM signal is stronger, we decided to patch the `dji_verify` binary to bypass the signature check and directly perform firmware package decryption.

From a high level, the entire process consists of the following steps:

1) Generate a modified firmware package (although during the first scan phase we use a fixed one).
2) Copy the file to the drone via ADB.
3) Execute the patched `dji_verify`.
4) Capture the EM data and go back to step 1.

Step 3 puts the hardware and software needed to perform the power analysis together. Figure 10 shows part of the setup used to record the power leak. The EM probe is attached to the XYZ table and positioned on top of the Leadcore SoC, where cryptographic operations take place.

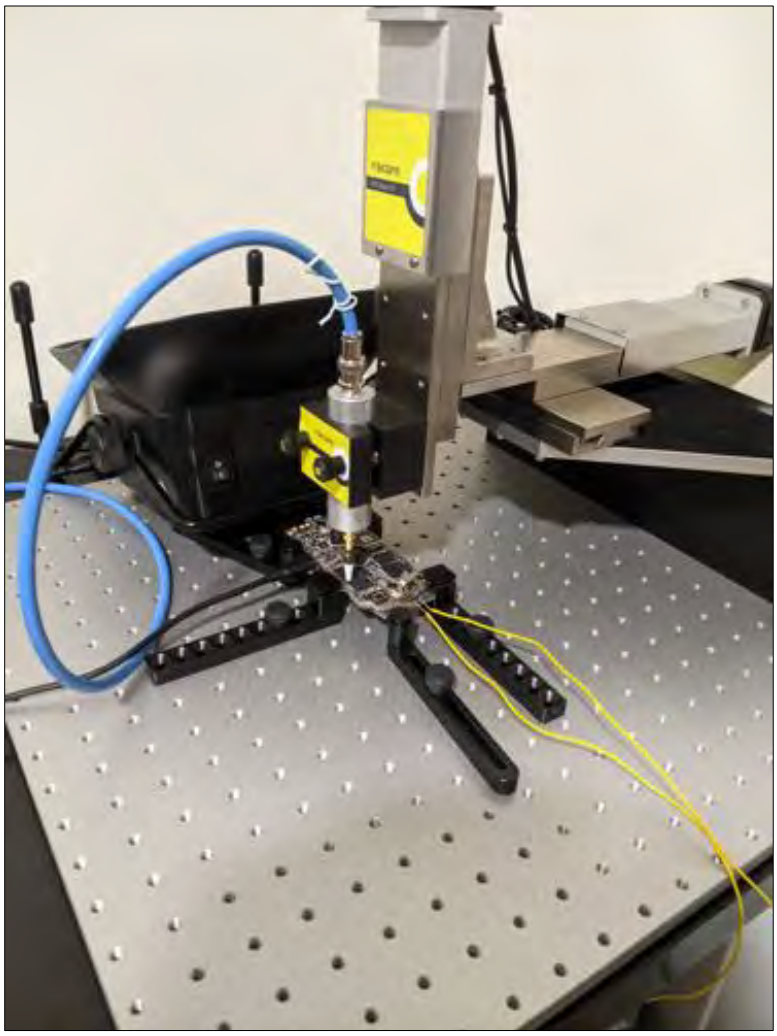


Figure 10: Side-channel setup ready to detect power leakage in the form of EM emanations

©2023 IOActive, Inc. All Rights Reserved.

One key feature of the tooling provided with Riscure's equipment was Inspector, a powerful software suite that implements all the statistical, mathematical analysis and data transformation required to perform SCA.

The only missing parts are those required to interact with each of the different in-scope device. For this project, IOActive developed Java code (included in Appendix B) that takes input arguments, such as repetitions, and coordinates launching the target binary. Inspector will take care of positioning the XYZ table, reading the EM signals, and plotting the results.

At this stage, the first goal was to find an area with a strong EM signal so we can place the probe and record thousands of traces to get enough information to extract the key.

After the location with strongest signal was identified, we moved from the patched `dji_verify` binary to the pristine setup, which includes the firmware signature verification. We then began studying the feasibility of bypassing the signature check with using EMFI.

At this point, we moved from Inspector to FI Python (FIPy), a web-based framework provided by Riscure to help with the FI process. This is described in more detail in the Second Approach section of this document. Figure 11 is the plot produced by FI Spotlight, showing the results of this round of testing.

Figure 11: Successful bypass (red), undetermined (orange), reboot (yellow), and normal operation (green)

After several days of trials and data analysis, we found that the probability of a successful signature bypass was lower than 0.5%. This renders key recovery unfeasible, since hardware-based crypto engines like the one used by the Leadcore SoC require hundreds of thousands of traces to be collected in order to perform an SCA attack.

At this point we paused further effort into this first approach, analyzed the target again, and came up with a second attack scenario.

©2023 IOActive, Inc. All Rights Reserved.

Second Approach

There were several potential avenues for bypassing security mechanisms using physical attacks, but most of them required modifications to the PCB. We typically would have used EMFI with the hopes of enabling JTAG to achieve debugging capabilities; however, we chose the path proposed in a research paper from Riscure[8] instead.

Riscure's researchers describe the different results they achieved while applying FI to the target SoC. There was one interesting outcome applicable to IOActive's research, where the glitches caused the processor to modify the content of destination registers during memory operations.

If the above results could be reproduced, we could potentially execute our own payload through the update process. This implies the following two requirements:

1) We have the ability to provide a custom payload to the target system. This we already know holds true from our work using the First Approach.
2) The process will copy memory from one location to another. Also from our work on `dji_verify`, we know the provided payload is first copied into memory to calculate the cryptographic signature to see if it matches the signed one.

In this new attack path, we attempted to demonstrate that gaining code execution using EMFI was plausible on the target device. A high-level overview of the process is as follows:

1) The update process copies the attacker-controlled payload from one memory location to another.
2) While the copy is taking place, generate a glitch with the goal of changing the behavior of the instructions being processed.
3) Analyze the result to determine what type of instruction modification has taken place.

To continue with this approach we needed to complete the following tasks:

1) Generate a firmware package that the `dji_verify` program would process.
2) Obtain a stack trace and register information to simplify the process of understanding whether an attack was successful. We used GDB for this step; this was one of the perks of having a controllable environment.
3) Setup the Riscure's glitching gear and adapt FIPy to work with our target.

To accomplish task 1, we modified the package header and added a file containing 10MB of 0x41 bytes as the payload, as described in the First Approach. This helps in evaluating the actual exploitability of the injection when analyzing the GDB output.

One constraint with the EMFI setup is finding an accurate hardware trigger. The density of components and the multi-layer PCB make it challenging to find the correct signal without risking the destruction of samples (one of the constraints for this project). To overcome this issue we relied on timing, based on the moment we received information through the ADB output; however, this approach reduces the repeatability of the injection, making it harder to find a successful hit.

[8] https://www.riscure.com/publication/controlling-pc-arm-using-fault-injection/

©2023 IOActive, Inc. All Rights Reserved.

Figure 12 shows the setup we used for EMFI to complete task 3. The EM probe is attached to the XYZ table and the Spider tool is connected to the computer that provides the timing for the trigger. The oscilloscope is used to verify we are injecting the correct pulse shapes at the right time, and its output is shown in Figure 13.

Figure 12: Setup used for FI during the second approach

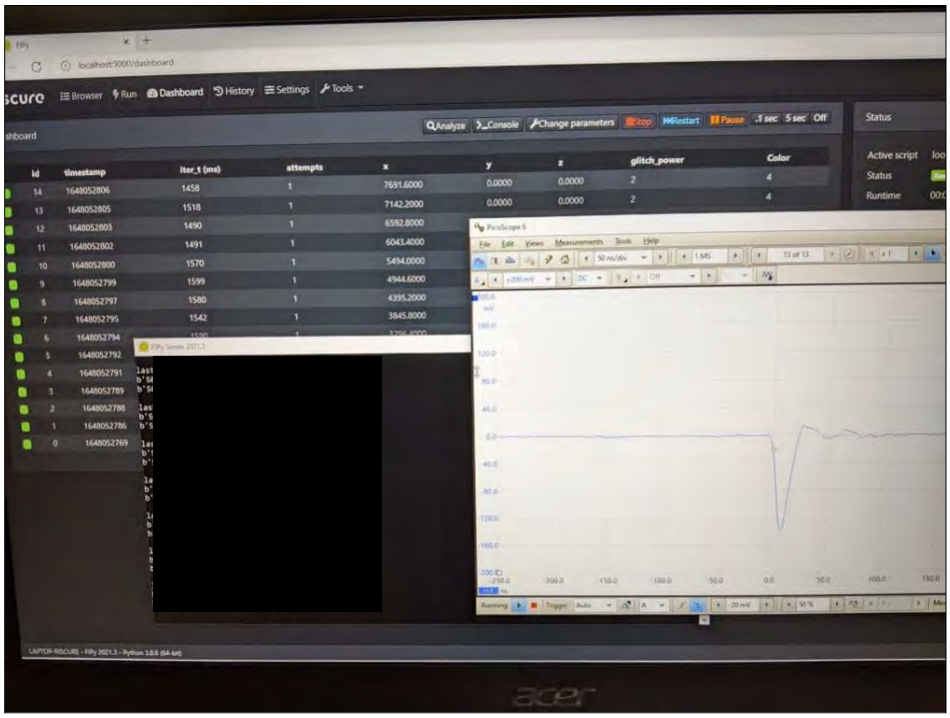


Figure 13: FIPy application alongside the oscilloscope showing the oscilloscope recording

©2023 IOActive, Inc. All Rights Reserved.

There is one difference between working with Inspector and FIPy: the former provides a Java interface and the latter executes a Python script. This time we put together a Python script (included in Appendix C) that manages the interactions with the target device, asks the Spider tool to trigger a glitch, and stores the results.

Once the whole setup was working flawlessly, we needed to run the test multiple times around the XY axis with different glitching strengths and timings until we identified a successful or potential error that could help us narrow down the surface area.

After identifying a small enough area, we modified the glitch's shape and timing until we observed a successful glitch. After numerous attempts, we got what we were looking for: the `dji_verify` program crashed, as shown in the following output.

```
Program received signal SIGSEGV, Segmentation fault.
0x2a002f44 in ?? ()
#0  0x2a002f44 in ?? ()
#1  0x2a000f5a in ?? ()
#2  0x2a0011de in ?? ()
#3  0x2a000b04 in ?? ()
#4  0xb6f9b42c in __libc_init () from /system/lib/libc.so
#5  0x2a00099c in ?? ()
Backtrace stopped: previous frame identical to this frame (corrupt stack?)
r0             0x41414141  1094795585
r1             0x41414141  1094795585
r2             0xbefff6dc  3204445916
r3             0x41414141  1094795585
r4             0x41414141  1094795585
r5             0xbefffbc4  3204447172
r6             0x85242d9a  2233740698
r7             0xbefff6f0  3204445936
r8             0xb6b4e008  3065307144
r9             0xb6b4e1e8  3065307624
r10            0xbefffad0  3204446928
r11            0x2d7160  2978144
r12            0xbefff6ec  3204445932
sp             0xbefff6c8  0xbefff6c8
lr             0xde  222
pc             0x2a002f44  0x2a002f44
cpsr           0x60000030  1610612784
```

Our payload appeared in several registers. After examining the code at the target address, we determined that we had hit a winning combination of timing, position, and glitch shape.

Figure 14 clearly shows the load instruction copying IOActive's data to registers R0 and R1. The GDB output above shows that registers R3 and R4 also ended up with controlled data.

```
ext:00002F40 loc_2F40                                    ; CODE
ext:00002F40                 MOV             R7, R2
ext:00002F42                 LDM             R7!, {R0,R1}
ext:00002F44                 STR             R0, [R3]
ext:00002F46                 STR             R1, [R3,#4]
ext:00002F48                 ADDS            R3, #8
ext:00002F4A                 CMP             R7, R12
ext:00002F4C                 MOV             R2, R7
ext:00002F4E                 BNE             loc_2F40
```

Figure 14: Instruction where the segmentation error took place

©2023 IOActive, Inc. All Rights Reserved.

Instead of copying data to the destination buffer, the following two instructions would copy R0 and R1 into a controlled address.

What most likely happened here is that we were able to modify the load instruction, and instead of reading only two registers, it ended up reading four. The original instruction encoding is:

```
LDM R7!, {R0, R1}                    0300B7E8
```

While the encoding of the instruction that would lead to the results we observed was:

```
LDM R7!, {R0, R1, R3, R4}            1B00B7E8
```

This meant that the glitch was able to modify one byte by flipping two consecutive bits and converting one instruction into another.

Having achieved this result, the next step would be to write a proper payload that turns this memory corruption into a code execution exploit. This could allow an attacker to fully control one device, leak all of its sensitive content, enable ADB access, and potentially leak the encryption keys.

©2023 IOActive, Inc. All Rights Reserved.

Future Work

In this paper we have described the materials and methods we used to successfully implement an attack on a UAV (drone) using EMFI techniques.

The goal of this project was to study the feasibility of such attacks on a complex and modern device. As we succeeded in proving that gaining runtime control is possible on the device while processing a firmware update, the next step is to apply the knowledge we acquired on another model from DJI with no previously known vulnerabilities.

To accomplish this task and create a reliable exploit within time and budget constraints, several elements must be improved:

1) **Hardware Trigger**: This is likely the one element that would most significantly reduce the time and complexity of the project. As mentioned throughout this paper, we have solely used a software, timing-based trigger, which makes it difficult to reliably know how to elicit a successful glitch.

   There are two different avenues to explore in order to implement the desired hardware trigger: (i) find an LED or available PCB trace from which we could detect a level change when a firmware update is being processed, or (ii) use Riscure's ICWaves to identify a power signature that coincides with the firmware upgrade package being processed.

2) **Integration**: Integrate all of the elements introduced in this report along with a hardware trigger and the firmware package being sent to the drone as it would work on actual firmware update.

3) **Payload**: Create a payload that could easily be checked to determine if the glitch was successful. There are different options, but one of the most obvious would be to enable ADB so we could check whether the USB connection is active from the host launching the attacks.

©2023 IOActive, Inc. All Rights Reserved.

Mitigations

An increasing amount of published research has shown promising results of applying EMFI to compromise a target system. This opens a new landscape of potential attacks and weaknesses in current and future products.

IOActive recommends that product developers implement EMFI countermeasures in their products, mitigating these attacks by implementing hardware and software countermeasures. Hardware countermeasures are very efficient at preventing EMFI attacks, but could be expensive and must be planned during the early design stages. Software countermeasures, on the other hand, can be added during the final stages of development, but might be less effective in mitigating certain attacks. Information on implementing EMFI countermeasures is publicly available in academic papers, conference presentations, and blog posts.

We also recommended evaluating EMFI countermeasures' effectiveness using a third-party lab like IOActive.

©2023 IOActive, Inc. All Rights Reserved.

Appendix A: Additional Information

The purpose of this Appendix is to provide additional information on relevant biographies, and present the glossaries for the figures, tables and equations utilized in this paper.

About IOActive

IOActive, a trusted partner for Global 1000 enterprises, provides research-fueled security services across all industries. Our cutting-edge security teams provide highly specialized technical and programmatic services including full-stack penetration testing, program efficacy assessments, and hardware hacking. IOActive brings a unique attacker's perspective to every engagement to maximize security investments and improve the security posture and operational resiliency of our clients. Founded in 1998, IOActive is headquartered in Seattle with global operations.

IOActive Labs Research Blog: http://blog.ioactive.com

About the Author

As the Director of Hardware Security for IOActive and a vulnerability researcher, Gabriel Gonzalez has completed hundreds of projects in the areas of reverse engineering, code review, and integrated hardware and software penetration testing. His primary focus is hardware and embedded system technologies, with specialized expertise in low-level attack vectors and fault-injection and side-channel analysis. Gabriel has engaged his research efforts in the fields of automotive, avionics, smart grid/utilities, SATCOM, banking devices, IoT, and many others.

Gabriel is actively engaged with the security research community, and has presented numerous original cybersecurity research projects at major conferences such as Black Hat Europe.

©2023 IOActive, Inc. All Rights Reserved.

Appendix B: Java Code

The following Java code takes input arguments, such as repetitions and coordinates, and launches the target binary.

```
/* copyright IOActive */

import static acquisition2.target.ErrorHandling.*;
import static com.riscure.common.data.SimpleVerdict.*;
import java.util.concurrent.TimeoutException;
import java.io.IOException;

// GUI parameter libraries
import com.riscure.osgi.annotation.Reference;
import com.riscure.beans.annotation.DisplayName;
import acquisition2.target.BasicSequence;
import org.osgi.framework.ServiceReference;
import com.riscure.osgi.legacy.Service;
import javax.validation.constraints.DecimalMax;
import javax.validation.constraints.DecimalMin;
import com.riscure.beans.constraints.DecimalStep;
import com.riscure.beans.annotation.Perturbation;
import com.riscure.beans.annotation.Presentation;
import com.riscure.beans.annotation.Unit;
import java.math.BigDecimal;
import java.beans.PropertyChangeListener;
import java.beans.PropertyChangeSupport;
import javax.validation.constraints.NotNull;

// @Service("DJI_MavicPro1_1st_try")
public class DJI_MavicPro1_1st_try extends BasicSequence {

    private DJI_MavicPro1_1st_trySettings settings = new
DJI_MavicPro1_1st_trySettings();

    @Override
    protected void init() {
        // Get all devices from hardware manager

        // Set default devices

        // If any error occurs the sequence will fail
        onError(FAIL);

        // Open all devices
    }

    @Override
    public void run() {

        // Set the default verdict to inconclusive
        verdict(INCONCLUSIVE);

        // Arm the measurement setup
```

©2023 IOActive, Inc. All Rights Reserved.

```
        arm();

        // Data generator will be selected and configured via Sequence GUI
settings,
        // data from the generator data can be fetched using following
method:
        byte[] generatorData = getGeneratorData(16); // get 16 bytes from
the data generator

        // From this point forward ignore errors
        onError(IGNORE);

        // Get the parameter values from UI
        System.out.println(String.format("Value of example parameter1 is:
%d", settings.getPerturParam().intValue()));
        System.out.println(String.format("Value of example parameter2 is:
%d", settings.getTargetParam().intValue()));

        //soft-trigger the measurement setup
        softTrigger();
        try {
            Process process =
Runtime.getRuntime().exec("C:\\Users\\ioactive\\platform-tools\\adb.exe
shell \"/cache/dji_verify -n 0100 -o /cache/test3 /cache/0x500.bin\"");
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

    @Override
    protected void onError(Throwable t) throws RuntimeException {
        if (t instanceof TimeoutException) {
            // Ignore
        } else {
            throw new RuntimeException(t);
        }
    }

    @Override
    public void close() {
        // Close all devices, note that closing does not mean powering
down
    }

    @Override
    public DJI_MavicPro1_1st_trySettings getSettingsBean() {
        return settings;
    }

    @Override
    public void setSettingsBean(Object settings) {
        this.settings = (DJI_MavicPro1_1st_trySettings) settings;
    }
```

©2023 IOActive, Inc. All Rights Reserved.

```
    @Override
    public boolean isDataGeneratorEnabled() {
        return true;
    }

    public static class DJI_MavicPro1_1st_trySettings {

        // Custom parameter examples
        @DisplayName("Example Parameter1")
        @Unit("Unit")
        @DecimalMin("0")
        @DecimalMax("10000000")
        @DecimalStep("4")
        @Presentation(slider = false)
        @Perturbation // This makes the parameter show up in the
Perturbation tab.

        private BigDecimal perturParam = BigDecimal.valueOf(4);

        public BigDecimal getPerturParam() {
          return perturParam;
        }

        public void setPerturParam(BigDecimal perturParam) {
          BigDecimal old = this.perturParam;
          this.perturParam = perturParam;
          pcs.firePropertyChange("perturParam", old, this.perturParam);
        }

        @DisplayName("Example Parameter2")
        @Unit("Unit")
        @DecimalMin("4")
        @DecimalMax("1000000")
        @DecimalStep("4")
        @Presentation(slider = false)

        private BigDecimal targetParam = BigDecimal.valueOf(4);

        public BigDecimal getTargetParam() {
          return targetParam;
        }

        public void setTargetParam(BigDecimal targetParam) {
          BigDecimal old = this.targetParam;
          this.targetParam = targetParam;
          pcs.firePropertyChange("targetParam", old, this.targetParam);
        }

        /*
         * Property change support
         */
        private PropertyChangeSupport pcs = new
PropertyChangeSupport(this);
```

©2023 IOActive, Inc. All Rights Reserved.

```
        public void addPropertyChangeListener(PropertyChangeListener
listener) {
            this.pcs.addPropertyChangeListener(listener);
        }

        public void addPropertyChangeListener(String propertyName,
PropertyChangeListener listener) {
            this.pcs.addPropertyChangeListener(propertyName, listener);
        }

        public void removePropertyChangeListener(PropertyChangeListener
listener) {
            this.pcs.removePropertyChangeListener(listener);
        }

        public void removePropertyChangeListener(String propertyName,
PropertyChangeListener listener) {
            this.pcs.removePropertyChangeListener(propertyName, listener);
        }
    }
}
```

©2023 IOActive, Inc. All Rights Reserved.

Appendix C: FIPy Controller Code

The following Python script manages interactions with the target device, asks the Spider tool to trigger a glitch, and stores the results.

```
from time import sleep, time
import pyvisa as visa
import subprocess
import sys
import os

####

from threading  import Thread

try:
    from queue import Queue, Empty
except ImportError:
    from Queue import Queue, Empty  # python 2.x

ON_POSIX = 'posix' in sys.builtin_module_names

def enqueue_output(out, queue):
    for line in iter(out.readline, b''):
        queue.put(line)
    out.close()


####

import serial
from fipy.parameters import (AttemptsParameter, FloatParameter,
IntParameter,
                             Parameters, SerialPortParameter)
from fipy.scriptutils import ResultColor, fipy_script
from spidersdk.chronology import Chronology
from spidersdk.spider import Spider

from fipy.parameters import *
from fipy.scriptutils import ResultColor, fipy_script
from spidersdk.chronology import Chronology
from spidersdk.spider import Spider
from fipy.transformutil import TransformUtil


PARAMETERS = Parameters(
    ('attempts', AttemptsParameter('Attempts')),

    ('xyz_scanner', SimpleXYZScanParameter('XYZ scanner')),
    ('glitch_power', IntParameter('Glitch Power', unit='%')),
    ('glitch_delay', IntParameter('Glitch delay', unit='ns')),
    ('spider_com_port', SerialPortParameter('Spider COM port')),

)
```

©2023 IOActive, Inc. All Rights Reserved.

```
EMFI_PULSE_AMPLITUDE = Spider.VOLTAGE_OUT1
EMFI_DIGITAL_GLITCH = Spider.GLITCH_OUT1
EMFI_GLITCH_VOLTAGE = 3.3
EMFI_GLITCH_LENGTH = 52e-9 #Glitch length parameter is fixed for the EMFI
Transient probe, we will use a 52ns pulse to trigger the EMFI probe;
output pulse length is 40ns with the 1.5mm probe tips
EMFI_GLITCH_DELAY = 0 # During this test we are not looking for the right
Delay just the Amplitud and position parameters
@fipy_script
def execute_script(util):
    counter = 0

    v = visa.ResourceManager()
    r = v.list_resources()
    if not r:
        print("NO POWER SUPPLY")
        exit(-1)
    dp832 = v.open_resource(r[0])

    # Inform the Inspector FI Python framework of our parameters, and
setup the database.
    util.set_termination_timeout(5)
    util.parameter_init(PARAMETERS)

    db = util.create_database_table('database.sqlite', 'database')
    util.add_to_cleanup(util.close_database)

    # Hardware initialization (see Spider API documentation for more
details)
    spider_com_port = serial.Serial()
    spider_com_port.port = PARAMETERS['spider_com_port'].value
    spider_com_port.open()
    util.add_to_cleanup(spider_com_port.close)
    spider_core1 = Spider(Spider.CORE1, spider_com_port)
    spider_core1.reset_settings()
    try:
        glitcher = Chronology(spider_core1)
    except IndexError as e:
        raise Exception(str(e) +
                        "\n\nDid you select the right COM port for Spider?
Is it powered on?")
    glitcher.forget_events()  # clear spider memory


    xyz_interface = util.get_xyz()
    tu = TransformUtil('table', xyz_interface.get_reference_points())

    # Initialize the digital glitch output
    glitcher.setVccNow(EMFI_DIGITAL_GLITCH, 0)
    # Initialize the pulse amplitude output
    glitcher.setPowerNow(EMFI_PULSE_AMPLITUDE, 0)

    transform = util.get_warping_tool()
```

©2023 IOActive, Inc. All Rights Reserved.

```
    do_reset = 0
    potential = 0
    pre_result = ''
    # Each iteration out paramaters receive new values here.
    for p in PARAMETERS:
        # Check if the stop or pause button has been pressed.
        if not util.process_commands():
            break


        # Record the timestamp for this iteration.
        tstamp = time()

        glitcher.setPowerNow(EMFI_PULSE_AMPLITUDE, p['glitch_power'])

        chip_pos = p['xyz_scanner']
        table_pos = tu.from_chip('table', chip_pos)
        xyz_interface.move_abs(table_pos.x, table_pos.y, table_pos.z,
hop_height=100)

        if reboot:
            localgdb = subprocess.Popen("C:\msys64\mingw64\bin>gdb-
multiarch.exe")
            \"/sbin/dji_verify -n 0100 -o /tmp/test
/cache/wm220_0100_v02.06.56.66_20171208.pro.fw.sig-fill-with-A 2>&1 \"",
stdout=subprocess.PIPE, start_new_session=True)

            q = Queue()
            t = Thread(target=enqueue_output, args=(localgdb.stdout, q))
            t.daemon = True
            t.start()

            resadb = ''

            localgdb.write("set architecture arm\n\r")
            localgdb.write("set exec-file /sbin/dji_verify")
            localgdb.write("target extended-remote localhost:9999")

            while True:
                try:
                    resadb = q.get_nowait()
                    print(resadb, cntloop)
                except Empty:
                    break
        exit(-1)

        glitcher.forgetEvents()

        glitcher.glitch(
            EMFI_DIGITAL_GLITCH,
            EMFI_GLITCH_VOLTAGE,
            p['glitch_delay'] / 1e9,
            EMFI_GLITCH_LENGTH)
```

©2023 IOActive, Inc. All Rights Reserved.

```
        glitcher.start()

        cntloop = 0
        resadb = ""
        while True and cntloop < 40:
            try:
                resadb = q.get_nowait()
                print(resadb, cntloop)
            except Empty:
                glitcher.forgetEvents()

                glitcher.glitch(
                    EMFI_DIGITAL_GLITCH,
                    EMFI_GLITCH_VOLTAGE,
                    0,
                    EMFI_GLITCH_LENGTH)

                glitcher.start()
                cntloop += 1
            else:
                print("-->", repr(resadb), table_pos.x, table_pos.y,
chip_pos.x, chip_pos.y)
                break

        # Block for 1 second or until Spider reaches final state.

        spider_timeout = glitcher.wait_until_finish(1000)

        if spider_timeout:
            color = ResultColor.PINK
        elif cntloop == 40:
            print("Maybe reboot?")
            dp832.write(":OUTP CH1,OFF")
            sleep(1)
            dp832.write(":OUTP CH1,ON")
            color = ResultColor.YELLOW
            sleep(20)
        elif resadb.find(b'[dji_image_verify') == 0:
            color = ResultColor.GREEN
        else:
            color = ResultColor.RED
            print("-->", repr(resadb), table_pos.x, table_pos.y,
chip_pos.x, chip_pos.y)

        adb.terminate()
        result = Parameters(
            ("id", counter),
            ("timestamp", int(tstamp)),
            ("iter_t (ms)", int((time() - tstamp) * 1000)),
            ("attempts", p['attempts']),
            ("x", chip_pos.x),
            ("y", chip_pos.y),
            ("z", chip_pos.z),
```

©2023 IOActive, Inc. All Rights Reserved.

```
            ("tx", float(table_pos.x)),
            ("ty", float(table_pos.y)),
            ("tz", float(table_pos.z)),
            ("glitch_power", p['glitch_power']),
            ("glitch_delay", p['glitch_delay']),
            ("Data", resadb),
            ("Color", int(color))
        )

        pre_result = result
        counter += 1

        # Send the results to the UI and the database.
        util.monitor(result)
        db.add(result)
```

©2023 IOActive, Inc. All Rights Reserved.