IEEE INFOCOM 2023 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS) | 978-1-6654-9427-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/INFOCOMWKSHPS57453.2023.10225819

Bewitching the Battlefield: Repurposing the MouseJack Attack for Crazyflie Drones

Narayana Murari Gowrishetty*, Sai Sree Laya Chukkapalli*, Anupam Joshi*
*Dept. of Computer Science & Electrical Engineering,
University of Maryland Baltimore County, Baltimore, USA
Email: {ngowris1, saisree1, joshi}@umbc.edu

***Abstract*—The Internet of Battlefield Things (IoBT) connects autonomous, uncrewed air and ground vehicles with wireless networks. It is a promising technology that has the potential to significantly enhance the operational effectiveness of military units by providing them with real-time information about the battlefield and enhancing their situational awareness. Autonomous assets such as drones and sensors can gather data from various sources and transmit it back to the command center, where it can be used to make informed decisions. However, the trustworthiness of the data provided by these autonomous assets depends on the systems' security and resilience to adversary attacks. Adversaries can potentially exploit vulnerabilities in the system to disrupt or manipulate the data. For example, adversaries can potentially hijack drones or disrupt the network bandwidth, causing Denial of Service (DoS) attacks or injecting fake data. In this paper, we demonstrate one such attack by showing how certain drone systems can be compromised by sniffing unencrypted wireless network channels to extract key information, create packets with chosen payloads, and send it to an unsuspecting host to affect its behavior. Specifically, we demonstrate that the initial steps of the MouseJack attack can be combined with packet reverse engineering to take control of Crazyflie drones.**

***Index Terms*—Internet of Battlefield Things, Security, Cyberattack, Drones**

I. Introduction

With the evolving capabilities of the Internet of Things (IoT), multiple sectors [1]–[6] have begun incorporating these innovative technologies into their infrastructure, allowing for enhanced capabilities that create smart infrastructure systems such as smart homes, smart cars, and smart grids. These advances have not only gained traction in civilian applications but have also started to impact military applications. Military planners are envisioning the future as one where soldiers and (semi)autonomous unmanned aerial and ground vehicles collaborate to carry out the mission objectives. Such systems that promote situational awareness and operational efficiency by integrating military assets with IoT are being called the Internet of Battlefield Things (IoBT) [7]. The IoBT allows a network of sensors, actuators, and uncrewed aerial and ground vehicles to gather operational data by continuously monitoring the battlespace and sharing information with the commander in control to make tactical decisions. For example, military personnel can remotely control uncrewed aerial vehicles such as drones for tactical reconnaissance. Moreover, the pervasive connectivity among the multitude of sensors associated with military combat equipment and soldiers enables data collection, information fusion, and sharing that provides supporting inferences for strategic decisions.

Several battlefield use cases, such as identifying enemy locations [8], responding to emergencies [9], and changing environmental conditions, indicate the potential of the IoBT network. However, IoBT networks are also vulnerable to cyberattacks similar to the IoT domain due to their weaker security design. In the past decade, numerous cyberattacks have been attempted against IoT infrastructures. The most prevalent attacks include the Stuxnet attack [10], the Mirai Botnet attack [11], and the DynDNS attack [12]. These attacks allowed attackers to gain unauthorized access to disrupt operations, steal data, or cause physical damage to the IoT infrastructures. IoT devices often provide large attack surfaces for attackers to exploit, where a couple of compromised sensors in the network can cause widespread damage to the entire network.

On the battlefield, adversaries may launch attacks to hijack drones, disrupt network bandwidth, cause Denial of Service (DoS) attacks [13], or inject fake data. The recent Russian attack on Ukraine [14] is another warfare example, where the Russians initially launched cyberattacks to disrupt or damage the infrastructure and services of the Ukrainian networks. These attacks can have severe consequences for the military unit relying on this information, as it could lead to incorrect decision-making or even loss of lives. In addition, adversaries could use such attacks to disrupt the operations of the military unit, potentially gaining an advantage in a conflict situation. Therefore, ensuring the security of the IoBT system should be of utmost importance.

In this paper, we show how to compromise small aerial drone systems often envisioned in drone swarm or reconnaissance tasks. Specifically, we show how the Crazyflie drone can be hijacked by combining well documented flaws in their RF system (by adapting the MouseJack attack [15]) and some packet level reverse engineering. First, we extract essential information and create packets with chosen payloads by sniffing unencrypted wireless network channels. Second, we control the operations of the Crazyflie drones as an adversary by utilizing packet reverse engineering technique in combination with MouseJack attack. For example, the Crazyflie drones fly over the intended area controlled by military personnel to share video feeds. However, we take control of the Crazyflie drone by preventing them from flying and disrupting the intended surveillance. Alternatively, the adversaries may transmit fake video feeds to the commander in control.

Similarly, several existing attack techniques [16], [17] show how to target other autonomous assets on the battlefield. Though the IoBT technology has the potential to enhance the capabilities of military units significantly, it is essential to carefully consider the security implications of using such systems. The results of this work indicate the need to develop effective security measures and robustness in the IoBT to prevent attacks and protect against potential vulnerabilities.

The rest of the paper is organized as follows. Section II describes the related work. In Section III, we provide details of the Crazyflie drones and explain the experimental setup. The steps for attacking the Crazyflie drone are mentioned in Section IV. Finally, we conclude in Section V.

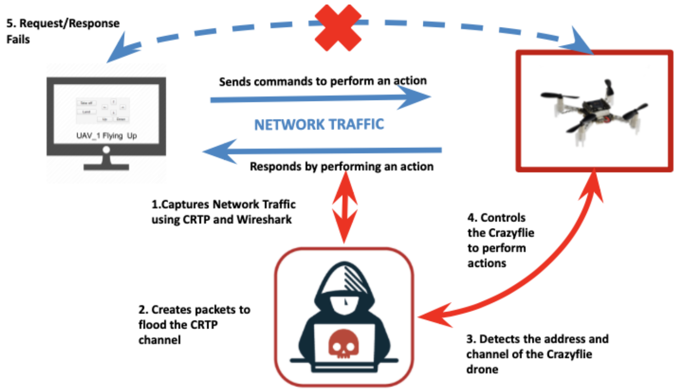

5. Request/Response Fails
Sends commands to perform an action
NETWORK TRAFFIC
Responds by performing an action
UAV_1 Flying Up
1.Captures Network Traffic using CRTP and Wireshark
4. Controls the Crazyflie to perform actions
2. Creates packets to flood the CRTP channel
3. Detects the address and channel of the Crazyflie drone

Fig. 1. Overview of our attack design.

II. Related Work

As military units and soldiers adopt more sophisticated and interconnected systems for their operations on the battlefield [18], they are increasingly relying on the Internet of Things (IoT) to facilitate communication and data sharing among various sensors and devices. While this has the potential to improve strategic decision-making and operational efficiency, it also creates new vulnerabilities and opportunities for cyberattacks. Recently, there has been a sharp uptick in cyberattacks targeting internet-connected devices [19]. These attacks can take many forms, including data breaches [20], that expose sensitive information, denial of service attacks [13], [21] that disrupt communication and operation, and website defacement [22] undermining the credibility and reputation of the organizations. Advanced and sophisticated attacks, such as advanced persistent threats (APTs) [23] to infiltrate and compromise a system's security over an extended period are another form of attack. In order to effectively defend against these types of attacks, military units must implement robust cybersecurity measures and continuously monitor and adapt to evolving threats.

These attacks have the potential to cause significant economic and societal impacts. An example is the Colonial Pipeline cyberattack that occurred in May 2021 against Colonial Pipeline [24]. This company operates a pipeline system that transports gasoline, diesel, and other refined petroleum products across the eastern United States. The attack was carried out by a criminal group known as DarkSide, which is believed to operate out of Eastern Europe. The group used ransomware to encrypt the company's systems and demanded a ransom in exchange for the decryption key. As a result of the attack, Colonial Pipeline was forced to shut down its operations, leading to fuel shortages and price spikes in parts of the southeastern United States. The attack also had broader implications, as it highlighted the vulnerability of critical infrastructure to cyberattacks. A second example is the Ukraine power grid, [25] which was targeted in a series of cyberattacks that disrupted the power supply to hundreds of thousands of people in 2015. The attacks targeted the Industrial Control Systems (ICS) of the power grid, causing power outages and damaging equipment. The attacks were the first known instance of a cyberattack being used to disrupt a power grid on a large scale.

Similarly, a network bandwidth attack that could potentially disrupt communication and operations, as shown in our previous work [26], prevents the information flow captured to the commander in control. This could have severe consequences for the effectiveness and safety of military operations and the security and safety of the soldiers and civilians involved. Therefore, we developed a context-aware policy-driven framework that demonstrates resilience by sharing information as much as possible, even when the network bandwidth drops. Moreover, an adversary can gain access to multiple assets by compromising a few devices as the battlefield interconnects various types of sensors, devices, and systems that communicate with each other and share data over a wider area. For example, an attack on a military command and control system could potentially allow an adversary to access and exfiltrate sensitive information.

Attackers have previously utilized MouseJack attacks on wireless devices to steal information. For example, a framework named KeyJack was introduced by Fournier et al. [27] to eavesdrop on wireless devices for retrieving data and further performing injection attacks. In this paper, we perform a MouseJack attack on Crazyflie drones to disrupt the flying operations of the drone.

III. Design & Experimental Setup

The network-connected assets on the battlefield are often critical to the success of military operations, making them attractive targets for adversaries seeking to gain an advantage. Our work illustrates how an adversary targets the autonomous assets present on the battlefield by either causing physical damage or manipulating the transmitted data. Fig 1 illustrates the overall design of the attack scenario. In this section, we explain the main components of Crazyflie, the security vulnerability chosen to perform the attack on Crazyflie drones, and the attack vectors and their specifics.

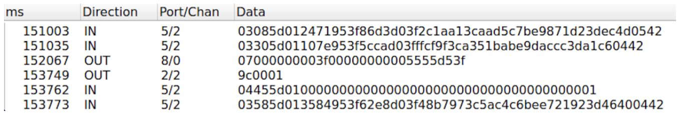
A. Overview of Crazy Real Time Protocol (CRTP)

Crazyflie drones and clients communicate over CRTP (Crazy Real Time Protocol) [28], where the protocol constructs the data packets such that they are routed easily to the various subsystems on the drone. The CRTP packet structure has three parameters, i.e. a port that ranges from 0-15 (4 bits), a channel that ranges from 0-3 (2 bits), and a payload that is a data buffer ranging up to 31 bytes in size. The protocol ensures that packets sent to a particular port are always in order, whereas reordering can take place while sending packets to different ports. Packets with the lowest port number have the highest priority in being routed to the subsystems. However, the communication protocol does not encrypt data which is a significant drawback. Moreover, CRTP is a connectionless protocol; except for a few subsystems in Crazyflie, like the log subsystem and radio link, the remaining are stateless. As a result, they are easily susceptible to eavesdropping, and the attacker can access the transmitted data between the controller and the drone.

B. Crazyradio PA 2.4 GHz Packet Analysis

In this section, we describe the packet analysis of the Crazyflie PA radio [29], which is based on a 2.4 GHz radio USB dongle and operates on a 2.4 GHz network. The packet analysis process involves analyzing data transmitted over the network to understand how it is utilized. We used various tools to perform packet analysis. These include packet capture tools to capture and record packets transmitted over the network for later analysis; protocol analyzers to decode and display the contents of packets in a more human-readable format; and network monitoring tools for real-time visibility into network traffic. The CRTP sniffer and Wireshark which we describe below, play a vital role in the analysis packets of Crazyflie drones.

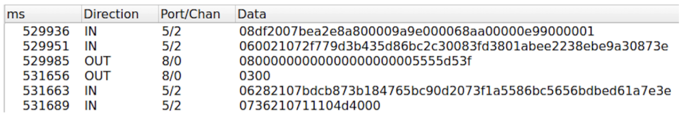
1) Bitcraze GUI console & CRTP Sniffer: In order to control the Crazyflie drone, we utilize the Bitcraze GUI console, which provides information about the telemetry and thrust value at each rotor. The Bitcraze GUI also offers a tool to view incoming and outgoing messages between the Bitcraze GUI client and the Crazyflie drone. Fig 2 displays the direction, port/channel, and data of the CRTP packet in chronological order. To capture the CRTP packets related to the drone's motion, we used the GUI controls to have the drone take off, navigate, and land. Simultaneously, we analyzed the CRTP sniffer tool logs and captured the flight control data to replicate it during our attack process. In Fig 3, we can see the outgoing packets responsible for the drone's landing. In addition, we also identify that the data values in Fig 2 for takeoff and Fig 3 for landing differ from each other.



ms	Direction	Port/Chan	Data
151003	IN	5/2	03085d012471953f86d3d03f2c1aa13caad5c7be9871d23dec4d0542
151035	IN	5/2	03305d01107e953f5ccad03fffcf9f3ca351babe9daccc3da1c60442
152067	OUT	8/0	07000000003f00000000005555d53f
153749	OUT	2/2	9c0001
153762	IN	5/2	04455d01000000000000000000000000000000000001
153773	IN	5/2	03585d013584953f62e8d03f48b7973c5ac4c6bee721923d46400442

Fig. 2. Outgoing packets captured by the CRTP sniffer tool during drone takeoff.

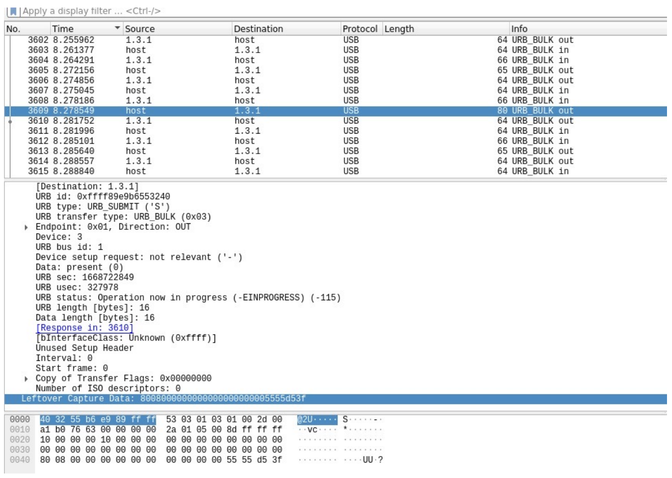
2) Wireshark: A shortcoming of the CRTP Sniffer tool described in Section III-B1 is that the data shown by the tool



ms	Direction	Port/Chan	Data
529936	IN	5/2	08df2007bea2e8a800009a9e000068aa00000e99000001
529951	IN	5/2	060021072f779d3b435d86bc2c30083fd3801abee2238ebe9a30873e
529985	OUT	8/0	0800000000000000000005555d53f
531656	OUT	8/0	0300
531663	IN	5/2	06282107bdcb873b184765bc90d2073f1a5586bc5656bdbed61a7e3e
531689	IN	5/2	0736210711104d4000

Fig. 3. Outgoing packets captured by the CRTP sniffer tool during the landing.

is in a human-readable form. We specifically want the data in the form of byte packets as it would be easy to manipulate the packets and transmit them over the wireless channels to perform injection attacks. To overcome this problem, we use Wireshark [30] to capture the incoming and outgoing messages transmitted over the Universal Serial Bus (USB) to which the Crazyradio PA dongle was connected. To perform this, we enable the "usbmon" Linux module to permit Wireshark to monitor the packets over the USB bus. We repeat the same drone exercise of take off, flying around, and landing, and capture the data packets on Wireshark instead of the CRTP sniffer tool. The packet data transmitted for the land command, as shown in Fig 4, correlates to the data seen in the CRTP sniffer in Fig 3. Once we have an actual message packet for flight command in byte format, we reconstruct the packets and transmit them using MouseJack as described in Section III-D1 and JackIt described in Section III-D2 tools.



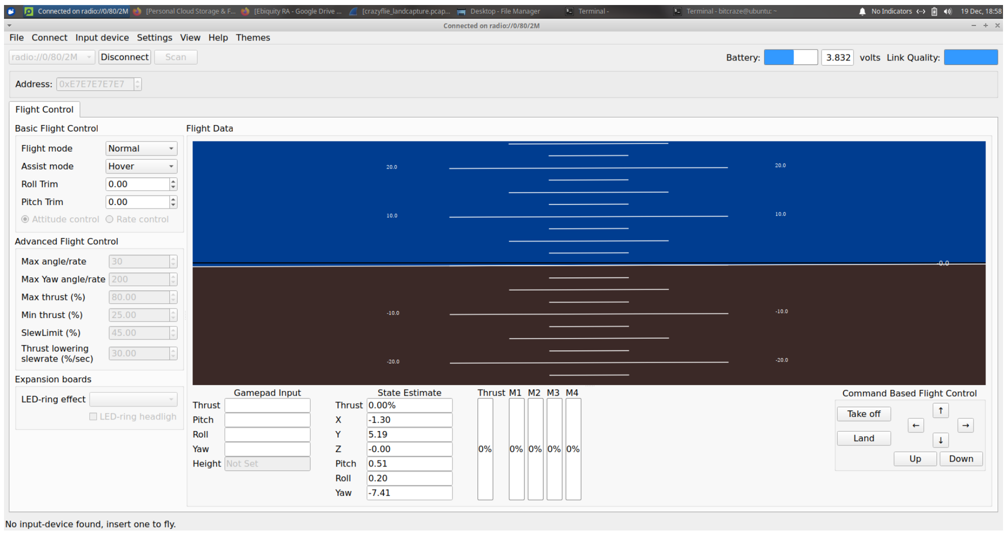
```
Apply a display filter … <Ctrl-/>
No.  Time       Source   Destination  Protocol Length  Info
3602 8.255962   1.3.1    host         USB              64 URB_BULK out
3603 8.261377   host     1.3.1        USB              64 URB_BULK in
3604 8.264291   1.3.1    host         USB              66 URB_BULK in
3605 8.272156   host     1.3.1        USB              65 URB_BULK out
3606 8.274856   1.3.1    host         USB              64 URB_BULK out
3607 8.275045   host     1.3.1        USB              64 URB_BULK in
3608 8.278186   1.3.1    host         USB              66 URB_BULK in
3609 8.278549   host     1.3.1        USB              80 URB_BULK out
3610 8.281752   1.3.1    host         USB              64 URB_BULK out
3611 8.281996   host     1.3.1        USB              64 URB_BULK in
3612 8.285101   1.3.1    host         USB              66 URB_BULK in
3613 8.285640   host     1.3.1        USB              65 URB_BULK out
3614 8.288557   1.3.1    host         USB              64 URB_BULK out
3615 8.288840   host     1.3.1        USB              64 URB_BULK in

    [Destination: 1.3.1]
    URB id: 0xffff89e9b6553240
    URB type: URB_SUBMIT ('S')
    URB transfer type: URB_BULK (0x03)
  ▸ Endpoint: 0x01, Direction: OUT
    Device: 3
    URB bus id: 1
    Device setup request: not relevant ('-')
    Data: present (0)
    URB sec: 1668722849
    URB usec: 327978
    URB status: Operation now in progress (-EINPROGRESS) (-115)
    URB length [bytes]: 16
    Data length [bytes]: 16
    [Response in: 3610]
    [bInterfaceClass: Unknown (0xffff)]
    Unused Setup Header
    Interval: 0
    Start frame: 0
  ▸ Copy of Transfer Flags: 0x00000000
    Number of ISO descriptors: 0
  Leftover Capture Data: 8008000000000000000000005555d53f

0000  40 32 55 b6 e9 89 ff ff  53 03 01 03 01 00 2d 00   @2U····· S·····-·
0010  a1 b0 76 63 00 00 00 00  2a 01 05 00 8d ff ff ff   ··vc···· *·······
0020  10 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  80 08 00 00 00 00 00 00  00 00 00 00 55 55 d5 3f   ········ ····UU·?
```

Fig. 4. Wireshark captures drone's land command packet data.

C. Crazyflie Python Library (cflib)

While the Crazyflie drone was airborne, an additional controller client using the Python library was needed to take over the drone and access the drone's address and channel information. The cflib [1] library is responsible for the motion of the Crazyflie drone, having classes such as *MotionCommander* and *Commander*. *MotionCommander* is the class that abstracts the calculation of telemetrics such as yaw, velocity, and coordinates in 3D space and provides the functions takeoff, land, forward, backward, left, and right, as shown in Fig 5. Another necessary functionality of the *MotionCommander* is

[1]https://github.com/bitcraze/crazyflie-lib-python



```
radio://0/80/2M
Battery: 3.832 volts  Link Quality:
Address: 0xE7E7E7E7E7

Flight mode            Normal
Assist mode            Hover
Roll Trim              0.00
Pitch Trim             0.00
(●) Attitude control   ( ) Rate control

Max angle/rate         30
Max Yaw angle/rate     200
Max thrust (%)         80.00
Min thrust (%)         25.00
SlewLimit (%)          45.00
Thrust lowering
slewrate (%/sec)       30.00

LED-ring effect
[ ] LED-ring headligh

Gamepad Input          State Estimate
Thrust                 Thrust  0.00%
Pitch                  X       -1.30
Roll                   Y       5.19
Yaw                    Z       -0.00
Height  Not Set        Pitch   0.51
                       Roll    0.20
                       Yaw     -7.41

Thrust 0%  M1 0%  M2 0%  M3 0%  M4 0%

No input-device found, insert one to fly.
```

Fig. 5. Bitcraze GUI to control Crazyflie drone.

to keep track of the state of the Crazyflie, i.e. the position in 3D space. *MotionCommander* class takes the Crazyflie instance as an attribute and maintains other attributes like is_flying flag and a SetPointThread. Since this class is a context manager, the setup, trigger of threads, and resetting of the position estimator happens when the context is created through the __enter__ method. The SetPointThread has an infinite running while loop, and a queue into which the velocity coordinates and yaw_rate are pushed whenever a user interacts with the functions such as take_off, move forward by 0.3 meters. In addition, *MotionCommander* class also calculates the velocity along each axis in x, y, and z directions and the yaw_rate to push into the SetPointThread.

The *Commander* class has various methods which convert parameters like velocity set points, yaw rate, roll, pitch, and thrust into CRTP packets and send them to the Crazyflie drone. In order to take over an already flying Crazyflie, there is a requirement to create a modified version of the *Motion-Commander* class, remove the context manager functionality, execute the SetPointThread in the __init__ method and set the is_flying flag to True. Since the cflib library has GNU General Public License, we were able to modify the source code of classes responsible for the flight control of drones. We could leverage existing code and use the available motion control methods to take control of the Crazyflie drone. However, the drone's address and channel are unavailable in a real-world scenario. To retrieve these details, we utilized the MouseJack with JackIt. Furthermore, developing a simple Python GUI client by integrating both modules to gather the address and channel number will help control the drone after hacking.

D. Overview of MouseJack and JackIt

1) MouseJack: In 2016, Bastille announced an open vulnerability called MouseJack [15], which still affects millions of devices, especially wireless mice and keyboards from most popular manufacturers like Amazon, Logitech, and Microsoft, which run on Radio Frequency (RF) channels. Bastille's team devised a way to sniff the network by sweeping a list of channels and decoding the enhanced ShockBurst packets picked up by the dongle. One of the affected dongles is the one that uses nRF24L transceivers. The nRF24L transceivers use a star network configuration where each node can simultaneously transmit and receive a maximum of six RF addresses. As a result, the dongle communicates with multiple wireless devices simultaneously. However, nRF24L chips do not officially support packet sniffing. In 2011, a pseudo-promiscuous mode [31] documented by Travis Goodspeed made it possible to sniff a subset of packets transmitted by other devices.

Our work identified that the Crazyradio PA, an amplified nRF24L-based USB dongle which controls the open-source Crazyflie drone, is vulnerable to the MouseJack exploit. The MouseJack[2] codebase has GNU General Public License v3.0, where the team provided research tools in Python scripts for scanning and firmware flashing of various RF dongles, one of which is the Crazyradio PA dongle. We utilize the firmware flash script provided by the Crazyflie team to flash the firmware of the Crazyradio PA dongle to enable wireless packet sniffing and injection, along with python scripts to sniff and scan the wireless channels around us. The modification of the Crazyradio PA firmware to add support for pseudo-promiscuous mode aids in packet sniffing and injection functionality. Therefore with the modified firmware of the Crazyradio PA dongle, we can get the functionality to scan the network, discover vulnerable devices, and communicate with the Crazyflie drone by flashing the dongle firmware.

2) JackIt: JackIt [3] is a partial implementation of MouseJack by Phikshun and Infamy. Here, the MouseJack scans and detects wireless devices, develops code to parse ducky scripts, converts them into keystrokes according to each keyboard

[2]https://github.com/BastilleResearch/mousejack
[3]https://github.com/insecurityofthings/JackIt

manufacturer, and performs keystroke injection. For our work, we utilize JackIt for wireless keystroke injection for keyboards from Logitech, Microsoft, and Amazon once the dongles are exploited, using MouseJack to detect the devices. We altered the code to work for drones in order to perform packet injection, and Denial of Service (DoS) attacks by channel flooding, and further integrate it with the Crazyflie Python library described in III-C to perform the attack instead of the keystroke injection.

E. Attack Vectors

1) Packet Injection: Our goal is to capture packets responsible for the motion control of the drone, as we plan to replay the captured packets to the drone to observe its actions, construct message packets similar to the ones we have captured, and send them to the drone from the attack system using a flashed Crazyradio PA dongle. However, we did not notice any discrepancies in the drone's flight. Therefore, we change our attack vector and attempt to perform a DoS attack by flooding the channels with messages to distort the network bandwidth and hinder the regular operation of the drone, but this did not affect the drone.

2) Denial of Service (DoS) Attack: After performing packet injection and not observing any response from the drone, we plan to perform a DoS attack by flooding the channels with CRTP packets and preventing communication between the drone and the controller. To achieve our goal, we modify the JackIt attack methods to flood the discovered channels repeatedly with CRTP message packets. However, we found no noticeable difference in the drone's movement; we could still control it from the original controller.

3) Takeover of Crazyflie drone: As discussed about cflib in Section III-C, we utilize the Python library to create scripts and connect to the Crazyflie drone. To connect to the drone, we need the URI to have Crazyflie's drone address, radio channel number, and transmission bit rate, as seen in Fig 6.

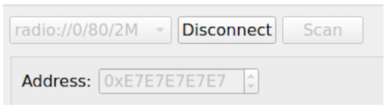


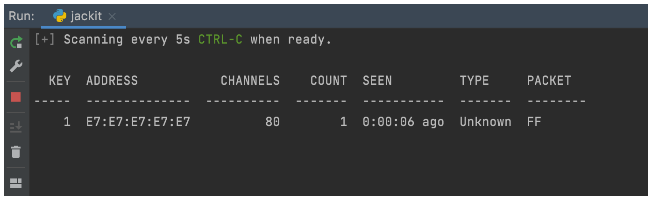
Fig. 6. Details of Crazyflie for URI construction.

But by using the original library, we cannot connect to a Crazyflie drone which is already airborne due to concurrency and locks on conditional variables. Hence, we modify the *MotionCommander* class from the library by removing the context managing ability, changing the flying flag to True, and executing the SetPointThread during initialization. The details we see in Fig 6 show how we obtain the address and channel number by executing the JackIt Python script. Once we identify the device to be attacked, we create a Crazyflie instance using the new URI generated from the address and channel number obtained and use the modified Motion Commander to take control of the drone.

IV. Attack Demonstration on Crazyflie Drones

In this section, we mention the steps involved in disrupting the operations of the Crazyflie drones from an adversary's perspective. Following the experimental setup described in Section III, we list the following steps to perform the attack.

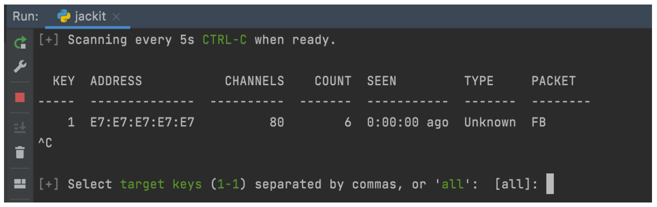
1) We use the Crazyradio PA dongle flashed with NRF Research firmware and a modified version of JackIt Python script for scanning the wireless channels to detect activity and find the addresses of devices vulnerable to wireless packet injection. As shown in Fig 7, we detect the address and channel number of the flying Crazyflie drone, which has the same values as Fig 6.



```
[+] Scanning every 5s CTRL-C when ready.

  KEY  ADDRESS          CHANNELS   COUNT  SEEN         TYPE     PACKET
-----  --------------   ---------- -------  -----------  -------  --------
    1  E7:E7:E7:E7:E7           80        1  0:00:06 ago  Unknown  FF
```

Fig. 7. Detected address and channel number of the Crazyflie.

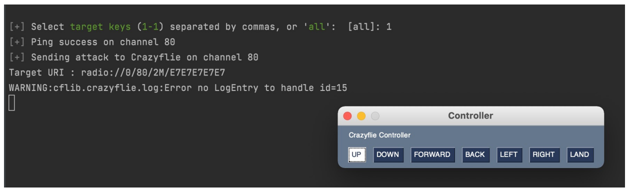
2) Once we identify the address and channel of the Crazyflie drone using the JackIt Python script, we proceed to the next phase, where we attack and take over the drone. To start the attack, we need to select the device number from the list of seen devices, as shown in Fig 8.



```
[+] Scanning every 5s CTRL-C when ready.

  KEY  ADDRESS          CHANNELS   COUNT  SEEN         TYPE     PACKET
-----  --------------   ---------- -------  -----------  -------  --------
    1  E7:E7:E7:E7:E7           80        6  0:00:00 ago  Unknown  FB
^C

[+] Select target keys (1-1) separated by commas, or 'all':  [all]:
```

Fig. 8. GUI console indicating an option to select one of the detected Crazyflie devices

3) After the detection phase, we use the cflib library, a native open-source Python library described in Section III-C provided by Bitcraze, to connect with the Crazyflie drones programmatically.
4) In particular, we use the asynchronous mode of the cflib where the Crazyflie Python instance connects to a stock Crazyradio PA dongle. In addition, the device address and RF channel discovered in the detection phase aids in URI construction.
5) Further, we utilize the modified version of the *MotionCommander* class from the cflib library and remove the synchronous connection features, bypass the connection setup, and initialize calibration to control the Crazyflie instance.
6) Now, we select a target device to attack, and a URI is automatically created behind the scenes. A Python GUI pops up on the screen. Using this, we can hijack and take control of the drone, as shown in Fig 9.



```
[+] Select target keys (1-1) separated by commas, or 'all':  [all]: 1
[+] Ping success on channel 80
[+] Sending attack to Crazyflie on channel 80
Target URI : radio://0/80/2M/E7E7E7E7E7
WARNING:cflib.crazyflie.log:Error no LogEntry to handle id=15
```

Fig. 9. Python GUI to control the hacked Crazyflie drone.

V. Conclusion

The Internet of Battlefield Things (IoBT) integrates the Internet of Things with military assets to support various military operations. These interconnected devices play an essential role in achieving mission-critical objectives by aiding the commander in control to continuously monitor various events on the battlefield and take necessary action. However, an adversary can potentially compromise these devices, which are prone to attacks, and gain access to sensitive information. Therefore, in this paper, we illustrate how devices like drones connected to the internet to provide real-time information are susceptible to attacks. In particular, we demonstrate the MouseJack attack by compromising the Crazyflie drones. We take control of Crazyflie drones by utilizing packet reverse engineering to analyze the data transmitted and perform a MouseJack attack to disrupt the intended operations. This involves sniffing unencrypted wireless network channels to extract essential information and create packets with chosen payloads that can affect the behavior of an unsuspecting host. Through this demonstration, we highlight the importance of ensuring the security and resilience of the IoBT system to prevent such attacks and maintain the trustworthiness of the data provided by these systems.

Acknowledgment

This research was supported by U.S. Army Grant No. W911NF2120076.

References

[1] Min Zhang, Tao Yu, and Guo Fang Zhai. Smart transport system based on "the internet of things". In *Applied mechanics and materials*, volume 48, pages 1073–1076. Trans Tech Publ, 2011.

[2] Sofia Dutta, Sai Sree Laya Chukkapalli, Madhura Sulgekar, Swathi Krithivasan, Prajit Kumar Das, and Anupam Joshi. Context sensitive access control in smart home environments. In *Int. Conferences on Big Data Security on Cloud, High Performance and Smart Computing, and Intelligent Data and Security*, pages 35–41. IEEE, 2020.

[3] Bahar Farahani, Farshad Firouzi, and Krishnendu Chakrabarty. Healthcare iot. In *Intelligent internet of things*, pages 515–545. Springer, 2020.

[4] Sai Sree Laya Chukkapalli, Nisha Pillai, Sudip Mittal, and Anupam Joshi. Cyber-physical system security surveillance using knowledge graph based digital twins-a smart farming usecase. In *2021 IEEE International Conference on Intelligence and Security Informatics (ISI)*, pages 1–6. IEEE, 2021.

[5] Miao Yun and Bu Yuxin. Research on the architecture and key technology of internet of things (iot) applied on smart grid. In *2010 international conference on advances in energy engineering*, pages 69–72. IEEE, 2010.

[6] Sai Sree Laya Chukkapalli, Shaik Barakhat Aziz, Nouran Alotaibi, Sudip Mittal, Maanak Gupta, and Mahmoud Abdelsalam. Ontology driven ai and access control systems for smart fisheries. In *Proceedings of the 2021 ACM Workshop on Secure and Trustworthy Cyber-Physical Systems*, pages 59–68, 2021.

[7] Stephen Russell and Tarek Abdelzaher. The internet of battlefield things: the next generation of command, control, communications and intelligence (c3i) decision-making. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*, pages 737–742. IEEE, 2018.

[8] Han Shi, Hai Zhao, Yang Liu, Wei Gao, and Sheng-Chang Dou. Systematic analysis of a military wearable device based on a multi-level fusion framework: research directions. *Sensors*, 19(12):2651, 2019.

[9] Gabriel Martins Leal, Iulisloi Zacarias, Jorgito Matiuzzi Stocchero, and Edison Pignaton de Freitas. Empowering command and control through a combination of information-centric networking and software defined networking. *IEEE Communications Magazine*, 57(8):48–55, 2019.

[10] Ralph Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[11] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. Understanding the mirai botnet. In *26th USENIX security symposium (USENIX Security 17)*, pages 1093–1110, 2017.

[12] Nicky Woolf. Ddos attack that disrupted internet was largest of its kind in history, experts say, Oct 2016.

[13] Roger M Needham. Denial of service. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, pages 151–153, 1993.

[14] James Lewis. Cyber war and ukraine, Jun 2022.

[15] Marc Newlin. Mousejack: White paper, 2016.

[16] Jyoti Deogirikar and Amarsinh Vidhate. Security attacks in iot: A survey. In *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*, pages 32–37. IEEE, 2017.

[17] Ruchi Vishwakarma and Ankit Kumar Jain. A survey of ddos attacking techniques and defence mechanisms in the iot network. *Telecommunication systems*, 73(1):3–25, 2020.

[18] Nikhil B Gaikwad, Hrishikesh Ugale, Avinash Keskar, and NC Shivaprakash. The internet-of-battlefield-things (iobt)-based enemy localization using soldiers location and gunshot direction. *IEEE Internet of Things Journal*, 7(12):11725–11734, 2020.

[19] Alexa Hernandez. Cyber attacks on iot devices are growing at alarming rates — venafi, Aug 2021.

[20] Goran Vojković, Melita Milenković, and Tihomir Katulić. Iot and smart home data breach risks from the perspective of data protection and information security law. *Business Systems Research: International journal of the Society for Advancing Innovation and Research in Economy*, 11(3):167–185, 2020.

[21] Sina Sontowski, Maanak Gupta, Sai Sree Laya Chukkapalli, Mahmoud Abdelsalam, Sudip Mittal, Anupam Joshi, and Ravi Sandhu. Cyber attacks on smart farming infrastructure. In *2020 IEEE 6th International Conference on Collaboration and Internet Computing (CIC)*, pages 135–143. IEEE, 2020.

[22] Content Team. Web defacement attacks: 5 website defacement examples, Jul 2021.

[23] Zhiyan Chen, Jinxin Liu, Yu Shen, Murat Simsek, Burak Kantarci, Hussein T Mouftah, and Petar Djukic. Machine learning-enabled iot security: Open issues and challenges under advanced persistent threats. *ACM Computing Surveys (CSUR)*, 2022.

[24] Tsvetan Tsvetanov and Srishti Slaria. The effect of the colonial pipeline shutdown on gasoline prices. *Economics Letters*, 209:110122, 2021.

[25] Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 388:1–29, 2016.

[26] Sai Sree Laya Chukkapalli, Anupam Joshi, Tim Finin, and Robert F Erbacher. Capd: a context-aware, policy-driven framework for secure and resilient iobt operations. In *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications IV*, volume 12113. SPIE, 2022.

[27] Guillaume Fournier, Pierre Matoussowsky, and Pascal Cotret. Hit the keyjack: stealing data from your daily wireless devices incognito. *arXiv preprint arXiv:1610.05212*, 2016.

[28] Communication with the crazyflie. https://wiki.bitcraze.io/projects:crazyflie:crtp.

[29] Crazyradio pa. https://wiki.bitcraze.io/projects:crazyradiopa:index.

[30] Ulf Lamping and Ed Warnicke. Wireshark user's guide. *Interface*, 4(6):1, 2004.

[31] Travis Goodspeed. Promiscuity is the nrf24l01+'s duty, 2011.