PROTC: PROTeCting Drone's Peripherals through ARM TrustZone

Renju Liu
UCLA CS
rl@cs.ucla.edu

Mani Srivastava
UCLA CS
mbs@ee.ucla.edu

ABSTRACT

As of Mar 2017, the FAA (Federal Aviation Administration) has more than 750k registered drone users. Safety of drones is the most crucial issue while designing drones. Most prior research focuses on aspects of the drone piloting system, drone applications, and drone cyber security. However, there lacks a system level protection for drone's essential peripherals. Several rootkits such as motochopper show that a commodity operating system is not safe, and the OS kernel can be easily compromised, such that the malicious applications can take control of the drone. We propose a new mechanism PROTC to protect the essential peripherals from being maliciously accessed. The protection is abstracted through the feature of ARM TrustZone. PROTC implements a trusted computing block within ARM TrustZone that enforces secure access control policy for the essential protected peripherals of the drone. The hardware protection from ARM TrustZone ensures that the trusted computing block of PROTC that runs privileged instructions is isolated from drone OS. PROTC successfully shows that only authorized applications can access drone's protected peripherals.

1. INTRODUCTION

Drones are becoming pervasive and are heavily used in a variety of industries in recent years because of their flexibility and ease of use. For example, people use drones for search and rescue, to inspect oil pipelines, or to take photographs. FAA has projected that the sales of commercial Unmanned Aerial Vehicles will be roughly 2.7 million by 2020 [18]. With the increased usage of drones, the security issues of drones are becoming urgent. Drones can cause millions of dollars loss if crashing with an airplane[20, 11].

The software development for drone, including drone apps and drone operating systems, is still at an early stage. In the current drone market at the time of writing (Mar 2017), the most popular drone piloting systems are ArduPilot [7] and PX4 [26]. ArduPilot and PX4 support either embedded OS or real-time Linux based OS. Some drones [34] choose embedded OS such as NuttX as their drone OS. The biggest drawback of embedded system is that the control application and other applications share the same address space. Hence, a malicious application can easily take control of the drone by overwriting the control application code on memory. Another drawback of embedded systems is that it is hard for the user to install a third-party application. For example, if the drone user decides to install a data collector, he or she needs to rebuild the whole system stack. More and more commercial drone companies [17] and drone SoC vendors [23, 10, 28] adopt real-time Linux based OS as the drone OS. The advantage of RT Linux based OS is that control applications and other applications have their own virtual memory space, such that a malicious application cannot directly overwrite the code of control application on memory. Another advantage of using RT Linux based OS is that the third party programs such as drone image processing program [24] and data collecting programs [32] can be easily installed during runtime without recompiling the whole system stack. However, RT Linux based OS exposes vulnerabilities that could be maliciously used to control the drone and steal important private data by giving the kernel space privilege to malicious user space program [6]. Malicious programs with rootkits, such as motochopper [2], Vroot [5], are the examples of where a malicious user space program can compromise the kernel and hence control the drone.

Our proposed mechanism, PROTC, can be used against memory attacks that escalate the privilege of user space applications. PROTC has two major goals: 1) it ensures the drone's safety and important data's integrity even when the drone's OS is compromised; 2) it allows for the installation of third-party applications easily (i.e. high flexibility). PROTC utilizes ARM TrustZone technology to achieve these two goals. ARM TrustZone divides the instructions on ARM processor into two privileged blocks. Lower privileged instructions are executed in *normal world*, and higher privileged instructions are executed in *secure world*. The secure world programs can restrict the memory access from the normal world programs. In PROTC, we regard the RT Linux based drone OS as untrusted OS that resides in normal world, and we design a trusted computing block resides in secure world to ensure that the access from an untrusted OS is permitted by the user.

PROTC has three important components: *critical applications, normal world applications* and the *trusted computing block*. Critical applications are those that need to access the protected peripherals and are installed inside the drone's

DroNet'17, June 23, 2017, Niagara Falls, NY, USA

DOI: http://dx.doi.org/10.1145/3086439.3086443

controller denoted as the ground control station in our design. Normal world applications are installed in normal world and do not have privileges to access any protected peripherals. The trusted computing block inside secure world is composed of a decision maker and a command executer. The decision maker enforces the protected peripherals' access control policy so that only authorized applications can access the protected peripherals. The command executer executes the command sent from critical applications if it passes the policy checking algorithm inside decision maker.

We propose four micro-benchmarks to test our PROTC prototype on Raspberry Pi 3 that is used by the commercial drone SoC NAVIO2 [23] with Linaro RT Linux OS in normal world and OP-TEE [3] OS in secure world. These four micro-benchmarks stand for the authorized and unauthorized access to protected peripherals with a base micro-benchmark that directly accesses peripherals without the protections of PROTC. Our results show that the average overhead introduced by PROTC is 143ms and that PROTC successfully prevents unauthorized access to protected peripherals.

Our contribution is: We design a system level protection mechanism, PROTC, to ensure that the access to protected peripherals is granted by the user, and that all the unauthorized access to protected peripherals is rejected.

The rest of the paper is organized as follows: we first discuss the background and related work for drones and ARM TrustZone in section 2 and section 3. We elaborate on the existing vulnerability in Linux based OS and our motivations to this work in section 4. In section 5 we introduce our PORTC model. We present preliminary results in section 6. We conclude our work in section 7 and discuss the future direction of this work.

2. BACKGROUND

2.1 ARM TrustZone

PROTC uses ARM TrustZone [8] protection features to create an extra secure policing zone. ARM TrustZone contains two different privilege blocks as shown in Figure 1.

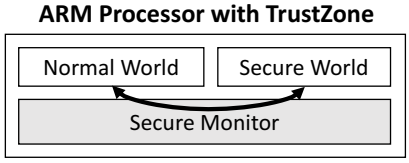

Figure 1: ARM TrustZone overview

Secure World is a privileged computing block that only runs privileged instructions. The secure world instruction is triggered when NS bit in the SCR register is not set. Once a program runs in secure world, it exposes the full access to the memories including those memory mapped peripheral registers. The secure world can define the memory regions that can only be accessed by privileged instructions.

Normal World runs all untrusted components such as untrusted operating system and its applications. The instructions inside normal world are unprivileged. Normal world instructions can only access the memory region if it is not marked as privileged memory region by secure world.

Secure Monitor is used to bridge secure world and normal world. The normal world programs cannot directly access the memory protected by secure world. However, when normal world program needs to get access from the memory region protected by secure world, it will trigger a secure monitor call (SMC), so that the context in normal world will be switched to secure world.

2.2 Open Portable Trusted Execution Environment (OP-TEE)

OP-TEE [3] is an open source project supported by several independent contributors and Linaro group. OP-TEE is aimed to provide the system support that runs on ARM TrustZone enabled SoCs.

OP-TEE contains a normal world OS and a secure world OS. Normal world OS is considered as untrusted OS while secure world OS is the privileged trusted OS. Normal world OS has a kernel driver that can use SMC to context switch to secure world OS. In PROTC, we use the APIs provided by OP-TEE to make the context switch between normal world and secure world.

3. RELATED WORK

Arm TrustZone. Prior research utilizing ARM TrustZone mainly focuses on smartphone domain. ARM TrustZone is used to monitor normal world kernel so that malicious code cannot be injected to kernel binaries [9], or to virtualize an OS apart from RTOS [30] . Brasser *et al.* [12] uses TrustZone to enforce smart devices to comply with privacy regulations. ARM TrustZone is also used to build trusted runtime on mobile devices [31, 36]. ARM TrustZone is also used to ensure data integrity from smartphone sensors [37, 15, 22]. In our work, we are not only targeting at protecting the integrity of data, but also providing a mechanism to prevent malicious access to drone's essential peripherals.

Drone Security. Most previous drone research focuses on deploying or localization aspects, such as reactive control for the drone pilot system [13], task cooperations [16], capturing cinema scenes [19], playing balls [27] and poles [14]. Few prior work in drone main focuses on security issue. AR. Drone described the vulnerability over the network communication because of the authentications on Telnet and FTP [29, 25]. Son *et al.* compromised drone sensors by generating resonance signals [35]. Sel *et al.* proposes a new framework to protect important data on drone delivery service path by enhancing the white-box cryptography [33]. However, none of the above work considers the vulnerabilities existing on drone's operating system. PROTC proposes a thorough protection for drone peripherals in system level.

4. MOTIVATIONS

Drone OS needs to ensure drone's control integrity against the memory attacks of malicious applications installed on drone and also allows the user to install third party applications easily. When considering both system security and easiness for users to install third party applications, embedded drone OS is not a good choice because the OS and the applications share the same address space and a malicious application can easily control the drone by directly writing to the memory mapped registers of protected peripherals such as actuators. Embedded OS is also hard to install third party applications because it requires the whole system stack to be recompiled each time.

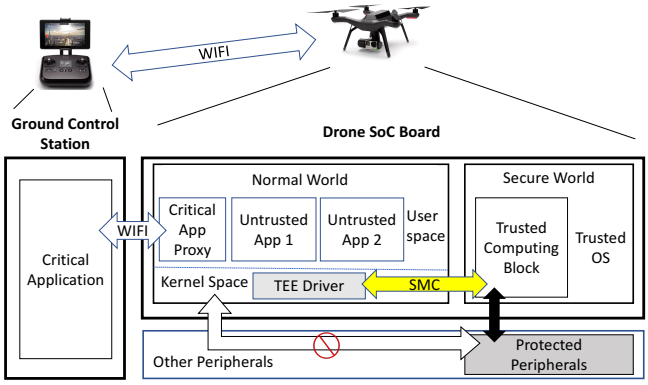

Figure 2: Overview of PROTC system design

The optimal choice for drone OS is to use real-time Linux based OS. Linux based OS is more secure because of the separation of address space, and it is also easy to install any third party applications. Drone OS needs to carry both drone's pilot program [7, 26] and other programs such as image processing programs [24, 1] and data collection programs [32]. In order to achieve such design, the security of drone OS might be reduced because piloting program and other programs share the same OS. Previous research shows that the user program can compromise the kernel through return-to-libc attack by overwriting the return address of a function stack [6].

A malicious application can utilize the rootkit [5, 4, 2] to compromise the kernel even when the OS is not rooted. Vroot [5] can change the destination of kernel function pointer to malicious user space code while maintaining its kernel privilege. Towelroot [4] and motochopper [2] can trick the kernel to change the kernel data in memory to escalate a user level process. Long *et al.* [21] shows that malicious program can easily escalate the user space privilege by wildcard injection, physical address attacks, etc in Linux. These attacks allow user processes to obtain kernel privilege so that they can have unrestricted access to all the peripherals.

All the attack methods and tools above demonstrate the vulnerabilities in current commodity general purpose OS such as Linux based OS. Our PROTC provides a protection mechanism to prevent the drone from being hijacked even when drone OS is compromised.

5. PROTC DESIGN

PROTC mechanism aims to provide a secure drone OS environment that prevents the drone from being maliciously controlled. The overview of PROTC design is shown in Figure 2. PROTC contains three essential components: *Critical Applications*; *Normal World Applications*; *Trusted Computing Block*. The critical applications are the applications that need to get certain access to protected peripherals, for example, navigation system. Normal world applications also named as untrusted applications might be proxy of critical application or independent application from third-party installed in normal world. The trusted computing block is the fundamental protecting block that enforces the security policy for the access to protected peripherals. The trusted computing block cannot be compromised because of TrustZone's hardware protection.

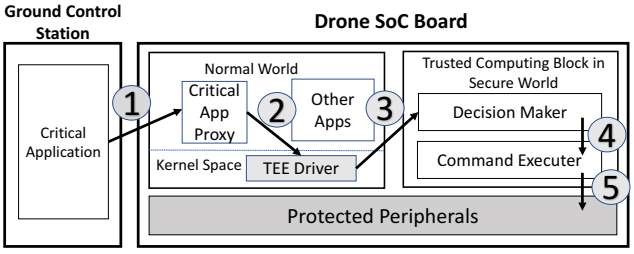

Figure 3: Procedure when critical application needs to access protected peripherals. Step①: Critical application sends the encrypted access request to its proxy in normal world. Step②: Critical app proxy passes the access request to TEE driver in normal world. Step③: TEE driver passes the access request through SMC to decision maker in trusted computing block. Step④: Decision maker decides whether to execute the command. Step⑤: Command executer executes the command if decision maker decides to execute.

5.1 Threat Model Assumptions

PROTC mechanism assumes the applications on ground control station in figure 2 are conditionally trusted under the restriction of access control policy described in section 5.6. For example, if application A on ground control station is authorized to access protected peripheral B, the behavior of A to access B is trusted. However, if application A is trying to access C that does not have access permission, this access is not trusted. The applications installed on drone SoC board are potentially malicious and not trusted. We assume the ground control station is secure and cannot be compromised. We also assume that the OS in normal world is not trusted. The trusted computing block of PROTC inside secure world are trusted and not malicious.

5.2 Trusted Computing Block

Trusted computing block is a secure guard program that resides in the secure world. It has two main features: 1) It enforces the access control policy described in section 5.6 so that only user approved access to protected peripherals can be permitted (*Decision Maker*). 2) When an access-authorized command is received from critical application, it needs to execute the command and return the results back to the critical application(*Command Executer*). As described in section 4, the OS in normal world might be compromised, hence the access to protected peripherals for normal world applications are revoked by trusted computing block, and these protected peripherals can only be accessed from trusted computing block due to ARM TrustZone protection.

The memory types of trusted computing block is both shared memory that can be read/written by normal world applications and non-shared secure memory that is only visible to trusted computing block. For all the sensitive data, such as the private key of trusted computing block that is used to sign protected sensor data sent to critical application, is stored in secure non-sharable memory that only trusted computing block has the privilege to access.

5.3 Critical Applications

Critical applications need part or full access to protected peripherals. The steps for critical applications to access pro-

tected peripherals are shown in Figure 3. One of critical applications is the drone control application that needs to access all the sensors and the actuators to fly the drone. In our model, critical applications could be malicious, for example, a sensor data collecting application is only given the access to sensors, but it contains malicious rootkit and tries to change the value of actuators to fly the drone.

Critical applications contain two parts. The first part is installed on ground control station that could be a drone controller or a computer on ground, and its main task is to send the access command to drone's trusted computing block. In the example of drone's pilot application, it sends the control command to drone according to the user's input. The second part of critical application is the proxy installed on the drone SoC board. Critical application proxy only transmits the information between ground control station and trusted computing block. However, some malicious critical applications could have malicious proxy that compromises drone normal world OS by rootkit described in section 4.

5.4 Normal World Applications

Applications in normal world are referred as untrusted applications. These applications could be independent applications or the proxy of critical applications. Untrusted applications are used to do tasks that do not require access to protected peripherals. For example, PX4D [24] can be installed as an untrusted application if camera is not protected. However, untrusted applications might be malicious. For example, if an image processing application contains rootkit that can escalate its privilege, the image processing application can directly write data to actuator memory mapped registers to control the drone without the implementation of PROTC.

5.5 Secure Communication Channel Establishment

When critical applications on ground control station need to send control commands to the drone, it will first establish a secure channel. The purpose is to assign the critical application an access number that will be further used to get authentication from trusted computing block to access the protected peripherals. The access code assigning algorithm includes the following two steps: 1) Critical Application → Trusted Computing Block: Pri_{CRI}(Access Request) (i.e. Critical application signs access request to trusted computing block); 2) Trusted Computing Block → Critical Application: Pri_{TCB} (Pub_{CRI} (Access Code)). Then critical application verifies the signature of trusted computing block and decrypt the message to obtain access code. We assume the program in trusted computing block has the public key of the critical application and the critical application has the public key of the program inside trusted computing block. We do not use the public key distribution process in PROTC, however, a valid and high efficient key exchange protocol could be implemented as SSL/TLS. Once the critical application obtains the access code, it uses the access code as part of the message while sending flight control command to the drone to reduce overhead trusted computing block needed when executing the command.

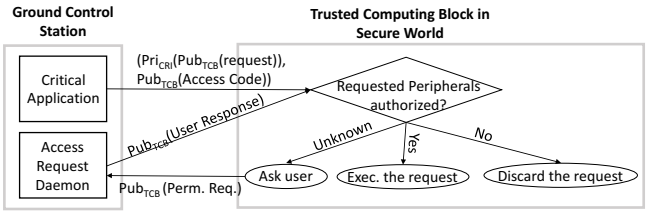

Figure 4: Access Control Decision Block Diagram. When critical application sends access request to trusted computing block, trusted computing block checks whether the access has already been authorized: if yes, execute the command; if no, discard the command; otherwise, send the access request back to user for approval.

5.6 Access Control Policy

The access control policy is enforced by PROTC decision maker to ensure the integrity of access to protected peripherals. The access permission to the protected peripherals is determined by the drone user. When an application on ground control station (i.e. critical application) wants to get access to protected peripherals, it will send a request through its proxy to trusted computing block. If it is the first time sending the access request, the trusted computing block will issue an access request inquiry message back to the access request daemon on ground control station, and the user will decide whether to give the access or not. Once the trusted computing block receives permission information from the user, it will store the information securely inside a hash table within the secure world. In order to reduce computation complexity, the trusted computing block generates a unique random access code for each application, where the random access code is hash-mapped to the application's public key and the protected peripheral access permission information. Once a critical application has access code, it only needs to send the information along with the access code to trusted computing block as (Pri_{CRI}(Pub_{TCB}(request)), Pub_{TCB}(access code)). An overview of access control decision process is shown in figure 4.

5.6.1 Sensors and actuators hierarchy

If the access request from an application to protected peripherals is denied, the remaining execution of the application might be problematic. For example, if an application needs to access IMUs to calculate the corresponding values sent to actuators, but the user denies this access mistakenly, the application might send dangerous command to actuators and crash the drone based on the problematic results from IMUs. We adopted a hierarchical access design to ensure a guaranteed access to lower level protected peripherals if the access to higher level is permitted.

In PROTC, the protected peripherals are divided into two hierarchical groups. All actuators that might affect drone's safety directly are grouped together as the top level. All other protected peripherals such as barometer and accelerometer are in a lower level. Moreover, the access privilege for actuators is one-for-all that if an application is allowed to access one actuator, it is automatically allowed to access all the other actuators and protected sensors. However, the access privilege for other protected peripherals is one-for-one that if an application has the access to cameras,

Name	Description
AUTH-FULL	The access to all protected peripherals is authorized.
AUTH-PART	The access to part of protected peripherals is authorized.
UNAUTH	The access to all protected peripherals is NOT authorized.
DI-ACCESS	Directly access to peripherals.

Table 1: Summary of micro-benchmarks used to evaluate PROTC design.

it does not guarantee the access to other sensors or actuators automatically.

6. EVALUATION

We build a prototype of PROTC to test and evaluate our design, and we propose four micro-benchmarks to evaluate PROTC design.

Hardware Setup. In our implementation, we choose Raspberry Pi 3 Model B board that is one of the cheapest ARM TrustZone supported development board as our prototype testbed platform. Raspberry Pi 3 Model B is also the motherboard of NAVIO2 [23] that is one of the most popular drone SoC board at the current market.

Software Setup. In PROTC, we follow the same NAVIO2 OS stack setups for our normal world. More specifically speaking, we choose Linaro that uses real-time Linux kernel in normal world to ensure real-time constraints. Trusted computing block is installed in kernel space of OP-TEE in secure world so that the time budget will be guaranteed. We also simulate the sensor data and actuator signals.

Micro-benchmarks. We propose four micro-benchmarks that are summarized in table 1. The first micro-benchmark is AUTH-FULL that has an authorized access permission to access all the protected peripherals. The second micro-benchmark is AUTH-PART that has authorized access permission to access some of the protected sensors only. The third micro-benchmark is UNAUTH that does not access permission to access any of the protected peripherals. The last micro-benchmark is DI-ACCESS, which is served as our baseline benchmark that directly accesses the peripherals from normal world OS through recompiling the whole system stack and disabling PROTC mechanism. These four micro-benchmarks represent all the possible cases that might happen in our system if the normal world OS is compromised. Through evaluating the micro-benchmarks, we prove that PROTC is against the memory attack that tries to control the drone by compromising the drone OS.

Prevention Analysis. AUTH-FULL and UNAUTH show that the protected peripherals can be accessed only when the user authorizes the access request. In PROTC, the access code is mapped to both critical application's public key and the permission to protected peripherals, so even if UNAUTH has AUTH-FULL's access code, UNAUTH cannot access the protected peripherals because the trusted computing block will use the public key the access code mapped (i.e. AUTH-FULL's public key) to verify the signature of the message. AUTH-FULL and AUTH-PART demonstrate that the permission given to higher level protected peripherals can be automatically propagated to lower level peripherals but not vice versa. In one of our experiment, we give AUTH-FULL the access privilege to actuators but not sensors, and due to the design of PROTC, AUTH-FULL still has the access privilege to protected sensors, and this avoids the potential bad values sent to actuators that might cause the crash of the drone. AUTH-PART is only given the access privilege for one protected sensor, so when it tries to access other protected peripherals, decision maker denies its access request. Our results show that even if the drone OS is comprised, the malicious application still cannot take control of the drone because it does not have permission to access protected peripherals, which also indicates that PROTC can be against memory related attacks such as buffer overflow attack and return-oriented programming (ROP) attack.

Micro-ben.	Average (ms)	Std. Dev. (ms)	Range (ms)
AUTH-FULL	143.15	0.16	0.64
UNAUTH	143.13	0.10	0.50
DI-ACCESS	0.34	0.02	0.07

Table 2: Statistics of overhead after running 100 times for each micro-benchmark(excluding WiFi latency)

Overhead. We run the benchmark DI-ACCESS, UNAUTH, AUTH-FULL 100 times to examine the overall overhead of our PROTC design, and the results are summarized in table 2. The average overhead introduced by PROTC in AUTH-FULL is 143ms. A further breakdown for AUTH-FULL overhead shows that the secure world session connection time from normal world applications occupies 21.49% of the time. The cryptographic algorithm (i.e. pub/pri key algorithm) of PROTC takes 61.17% of time, and the rest including sending and executing the commands from normal world to secure world takes 17.34%. The results of AUTH-FULL and UNAUTH tell that the significant overhead of PROTC in AUTH-FULL is the cryptographic algorithms.. The overhead of command executer inside trusted computing block is only about 200 μs. Because PROTC mechanism satisfies real-time constrains by adopting RT-Linux kernel in normal world and use kernel space in secure world, after we run the experiments 100 times, it shows that the range of AUTH-FULL is 0.501ms and the deadline while executing a command will be guaranteed within 144ms.

7. CONCLUSION REMARK

PROTC successfully demonstrates that a new drone system design based on ARM TrustZone technology can protect drone from being maliciously controlled. PROTC also shows that it is able to be against memory attacks on drone OS. A future direction of this work is to utilize trusted computing block in PROTC to determine whether to execute a command based the values of the sensors.

Acknowledgment

This research is funded in part by the National Science Foundation under awards CNS-1329755. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF, or the U.S. Government. We would also like to thank all the anonymous reviewers for their helpful comments.

8. REFERENCES

[1] https://event38.com/drone-data-management-system/.

[2] Motochopper. http://hexamob.com/how-to-root/motochopper-method/.

[3] Open portable trusted execution environment. https://www.op-tee.org/.

[4] towelroot. https://towelroot.com/.

[5] vroot. https://androidmtk.com/download-vroot.

[6] Return-oriented rootkits: Bypassing kernel code integrity protection mechanisms. In Presented as part of the 18th USENIX Security Symposium (USENIX Security 09) (Montreal, Canada, 2009), USENIX.

[7] ArduPilot. http://www.ardupilot.org/.

[8] ARM. Arm trustzone. https://www.arm.com/products/security-on-arm/trustzone.

[9] Azab, A. M., Ning, P., Shah, J., Chen, Q., Bhutkar, R., Ganesh, G., Ma, J., and Shen, W. Hypervision across worlds: Real-time kernel protection from the arm trustzone secure world. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (New York, NY, USA, 2014), CCS '14, ACM, pp. 90–102.

[10] Bebop, P. https://www.parrot.com/us/Drones/Parrot-bebop-2.

[11] Bloomberg. https://www.bloomberg.com/news/articles/2016-04-04/drones-are-the-new-threat-to-airline-safety.

[12] Brasser, F., Kim, D., Liebchen, C., Ganapathy, V., Iftode, L., and Sadeghi, A.-R. Regulating arm trustzone devices in restricted spaces. In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (New York, NY, USA, 2016), MobiSys '16, ACM, pp. 413–425.

[13] Bregu, E., Casamassima, N., Cantoni, D., Mottola, L., and Whitehouse, K. Reactive control of autonomous drones. In Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services (New York, NY, USA, 2016), MobiSys '16, ACM, pp. 207–219.

[14] Brescianini, D., Hehn, M., and D'Andrea, R. Quadrocopter pole acrobatics. In 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (Nov 2013), pp. 3472–3479.

[15] Colp, P., Zhang, J., Gleeson, J., Suneja, S., de Lara, E., Raj, H., Saroiu, S., and Wolman, A. Protecting data on smartphones and tablets from memory attacks. In Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 177–189.

[16] Dantu, K., Kate, B., Waterman, J., Bailis, P., and Welsh, M. Programming micro-aerial vehicle swarms with karma. In Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (New York, NY, USA, 2011), SenSys '11, ACM, pp. 121–134.

[17] DJI. http://www.dji.com/.

[18] FAA. Faa aerospace forecast fiscal year 2016 - 2036.

[19] Fleureau, J., Galvane, Q., Tariolle, F.-L., and Guillotel, P. Generic drone control platform for autonomous capture of cinema scenes. In Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (New York, NY, USA, 2016), DroNet '16, ACM, pp. 35–40.

[20] govtech. http://www.govtech.com/products/Analysis-Drones-Posing-Global-Security-Issues.html.

[21] II, M. C. L. Attack and defend: Linux privilege escalation techniques of 2016.

[22] Liu, H., Saroiu, S., Wolman, A., and Raj, H. Software abstractions for trusted sensors. In Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (New York, NY, USA, 2012), MobiSys '12, ACM, pp. 365–378.

[23] NAVIO2. https://emlid.com/introducing-navio2/.

[24] PIX4D. https://pix4d.com/.

[25] Pleban, J.-S., Band, R., and Creutzburg, R. Hacking and securing the ar. drone 2.0 quadcopter: investigations for improving the security of a toy. In IS&T/SPIE Electronic Imaging (2014), International Society for Optics and Photonics, pp. 90300L–90300L.

[26] PX4. https://dev.px4.io/.

[27] Ritz, R., Müller, M. W., Hehn, M., and D'Andrea, R. Cooperative quadrocopter ball throwing and catching. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (Oct 2012), pp. 4972–4978.

[28] Robotics, E. http://docs.erlerobotics.com/.

[29] Samland, F., Fruth, J., Hildebrandt, M., Hoppe, T., and Dittmann, J. Ar. drone: security threat analysis and exemplary attack to track persons. In Proceedings of the SPIE (2012), vol. 8301.

[30] Sangorrin, D., Honda, S., and Takada, H. Dual operating system architecture for real-time embedded systems. In Proceedings of the 6th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT), Brussels, Belgium (2010), pp. 6–15.

[31] Santos, N., Raj, H., Saroiu, S., and Wolman, A. Using arm trustzone to build a trusted language runtime for mobile applications. In Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (New York, NY, USA, 2014), ASPLOS '14, ACM, pp. 67–80.

[32] SenseFly. https://www.sensefly.com/applications/gis.html.

[33] Seo, S.-H., Won, J., Bertino, E., Kang, Y., and Choi, D. A security framework for a drone delivery service. In Proceedings of the 2Nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use (New York, NY, USA, 2016), DroNet '16, ACM, pp. 29–34.

[34] Solo, D. https://3dr.com/solo-drone/.

[35] Son, Y., Shin, H., Kim, D., Park, Y., Noh, J., Choi, K., Choi, J., and Kim, Y. Rocking drones with intentional sound noise on gyroscopic sensors. In 24th USENIX Security Symposium (USENIX Security 15) (Washington, D.C., 2015), USENIX Association, pp. 881–896.

[36] Sun, H., Sun, K., Wang, Y., Jing, J., and Wang, H. Trustice: Hardware-assisted isolated computing environments on mobile devices. In Dependable Systems and Networks (DSN), 2015 45th Annual IEEE/IFIP International Conference on (2015), IEEE, pp. 367–378.

[37] Yussoff, Y. M., and Hashim, H. Trusted wireless sensor node platform.