Regulating Drones in Restricted Spaces

Abhishek Vijeev, Vinod Ganapathy, Chiranjib Bhattacharyya
Department of Computer Science and Automation and Robert Bosch Centre for Cyber-Physical Systems
Indian Institute of Science, Bangalore-560012, India
abhishekvijeev@iisc.ac.in,vg@iisc.ac.in,chiru@iisc.ac.in

ABSTRACT

Commercial and end-user drones come equipped with a wide array of sensors. Unregulated use of such drones in public airspaces poses a serious threat to the privacy of citizens. We make the case for *restricted spaces* for drones, which are geographic areas for which a *host* can specify its privacy policies. *Guest* drones must prove to the host that they are in compliance with the host's policies before entering the restricted space. We then make the case for an information-flow control-based policy enforcement framework on drones, and sketch the design of a prototype framework atop the Robot Operating System (ROS).

CCS CONCEPTS

• **Security and privacy → Mobile platform security**; **Information flow control**;

KEYWORDS

Drones; privacy; restricted spaces; trusted hardware

ACM Reference Format:
Abhishek Vijeev, Vinod Ganapathy, Chiranjib Bhattacharyya. 2019. Regulating Drones in Restricted Spaces. In *The 20th International Workshop on Mobile Computing Systems and Applications (HotMobile '19), February 27–28, 2019, Santa Cruz, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3301293.3302370

1 INTRODUCTION

Commercial and end-user drones are becoming widely available. Such drones can be employed for a number of interesting and socially-beneficial use-cases, such as sensing, search and rescue, and product delivery. However, the wide availability of drones has also put a previously tightly-regulated resource, i.e., airspace, into the hands of commercial entities and end-users. We are already beginning to see an increasing number of cases where commercial drones can pose dire risks. Incidents involving "near-misses" between drones and aeroplanes are becoming increasingly common, with drones sightings being reported as high as 15,500 feet [34]. In August 2018, two DJI Matrice 600 drones were used to carry out an attack in Caracas during an address by the Venezuelan president at a military event [17]. London Gatwick airport was shut down for approximately three days in December 2018, causing major disruption, after suspicious drones entered the restricted airspace of the airport.

Regulatory bodies, such as the Federal Aviation Authority (FAA), and other law-enforcement agencies are actively seeking to evolve tighter rules for the shared use of airspace by commercial and end-user drones [3, 7, 8, 11, 13]. The impact of regulations on the frequency of such drone-related incidents will only become clear over time. However, we posit that even in a regime where rules regarding the shared use of airspace are clearly formulated and enforced, the future deployment of drones (*e.g.,* those expected to be used by Amazon's Prime Air) will pose a serious risk to the privacy of citizens. Most commercial drones come equipped with cameras and other advanced sensors that can record the environment around them. These sensors are *essential* features of drones, *e.g.,* video feeds and images captured from the drone are used for either autonomous navigation or remote human control of the drone. However, these very sensors can be misused to compromise privacy.

In this paper, we propose to address these issues by developing the vision of *restricted spaces* for drones. A restricted space is an area geographically demarcated by its *host*, within which the host expects *guest drones* to conform to its usage policies. We present the design of a framework via which the host can communicate its privacy policies to a guest drone, and ensure that the policies are enforced on the guest drone.

2 RESTRICTED SPACES FOR DRONES

We now develop our vision of restricted spaces for drones. The host of the restricted space may configure its security policies to suit its privacy needs. We consider three examples below:

- *Process-Images-Locally*: The host may require that any images or video feeds captured by the drone be processed locally within the drone and should not be transmitted over the network. The host may additionally require that the images not be stored in the drone's persistent storage (from where an attacker may recover them later). Such a policy can be applied to autonomous drones that have sufficient processing power to locally process images/video for navigation.
- *Blur-Exported-Images*: If the drone is manually controlled or requires its image/video feed to be processed by a back-end cloud server, then the host may require the image/video feed to be processed by a filtering service that blurs sensitive portions of the image/video (*e.g.,* faces and car registration plates) before the being transmitted to the cloud server. The filtering service could either be a trusted application running within the drone, or a cloud-based service controlled by the host. In the former case the host can leverage trusted hardware on the drone to establish the existence of a filtering service on the drone.

HotMobile '19, February 27–28, 2019, Santa Cruz, CA, USA

ACM ISBN 978-1-4503-6273-3/19/02…$15.00
https://doi.org/10.1145/3301293.3302370

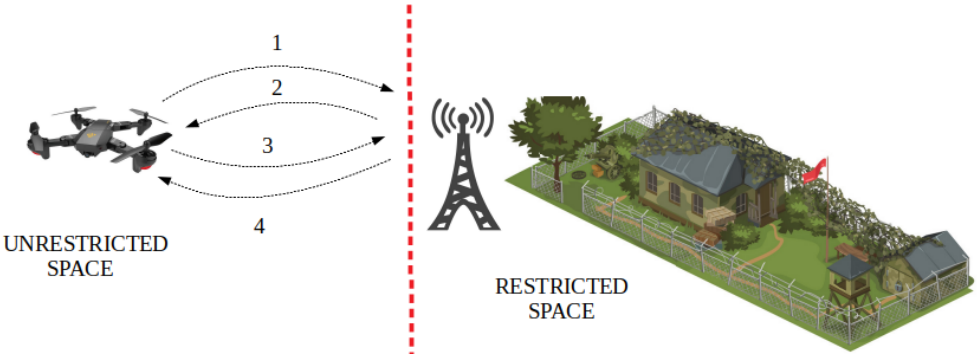

Figure 1: *Restricted spaces for drones.* (1) When a *guest drone* wishes to enter a *host's* restricted space, it presents its credentials to the host and requests its security policy; (2) The host responds with its security policy, which the guest drone then analyzes to determine if the policy is acceptable to it; (3) If the policy is acceptable, the drone enforces the policy on its software stack, and sends a cryptographic proof to the host that it is in policy compliance; (4) The host verifies the proof and authorizes the drone to enter.

• *Log-GPS*: The host may require the guest drone to only fly along pre-defined "drone lanes" when within its restricted space. The host can check this by requiring the drone to log its GPS feed during its stay in the restricted space, and to submit this GPS log for analysis as it leaves [20].

When a guest drone enters the restricted space from the outside, it must first "check-in" with the host to determine the host's security policy (see Figure 1). For instance, it could communicate with a Wi-Fi access point or tower controlled by the host. In a 5G or WiMax setting, for instance, the host could have communicated its policy to the 5G-provider, which tags the policy with the host's GPS markers. The drone can directly obtain the policy from the 5G-provider as it approaches the restricted space. During check-in, the drone presents its credentials (*e.g.,* its public key) to the host. The host sends its policy to the drone, which the drone can then choose to analyze to determine if it is acceptable.

If the guest finds the policy too restrictive, it can choose to reject the policy and not enter the host's restricted space. At this point, the host can take suitable action if the drone still chooses to enter its restricted space. The action taken depends on the nature of the host. For example, a defense establishment could choose to shoot down the drone; in contrast, a commercial establishment could choose to report the drone's credentials (or perhaps a close-up picture of the drone) to a regulatory body.

If the guest drone finds the policy acceptable, it applies the policy to the applications running on the drone. It then proves to the host that it is in compliance with the host's policies, and is then granted approval to enter the restricted space. To enable cryptographically-sound proofs, we assume that the drone is equipped with trusted hardware (see threat model below). Once it leaves the host's restricted space, it can "check-out" by choosing to resume execution without the host's policy restrictions. Depending on the host's policy, the drone may be required to submit some information (*e.g.,* its GPS log) during check-out, which the host can then post-process to check for compliance.

A drone will typically fly over multiple such restricted spaces during a single flight to its destination. It must conform to the policy restrictions of all the hosts enroute, or choose to take an alternative path to the destination.

Threat Model. We consider a threat model where hosts assume that guest drones are under adversarial control. Drones must submit their credentials and prove that they are compliant with the host's policies before they enter the restricted space.

To enable robust compliance proofs, we require that the drones be equipped with trusted hardware, such as the ARM TrustZone [4]. Such hardware is becoming increasingly available on commodity devices, and offers a hardware root of trust on the guest device. The hardware is endowed with a public/private key pair, and can produce digitally-signed attestations of the state of the software stack running on the guest drone. The guest's public key (together with its digital certificate) could also serve as the credential used by the host to identify the drone. The host verifies these attestations to determine the guest's security posture. We discuss the details of the attestation process in Section 3. We assume the existence of a regulatory authority that hosts can approach to report drones that enter the restricted space without complying with the host's policies.

Our threat model does *not* currently consider the following two important cases:

- Drones that are not equipped with trusted hardware.
- Covert use of drones.

Without trusted hardware on the drone, hosts do not have a mechanism to reliably identify the drone or verify that it is in compliance. We hypothesize that foolproof verification of the guest drone's security posture is impossible without trusted hardware. Fortunately, consumer devices are increasingly being equipped with ARM TrustZone, and indeed, future regulations may require that commercial drones have a reliable way to establish their identity, *e.g.,* akin to registration plates on vehicles.

A drone may covertly enter the restricted space and compromise the host's privacy. In cases where drones have a miniature form-factor, it may even be difficult for a host to determine that an unauthorized drone is present in its restricted space. Such covert use of drones poses a major risk to privacy, and new methods are needed to detect the presence of such drones.

① **Drone → Host**:	Pub_{drone} + certificate
	Host verifies certificate
② **Host → Drone**:	Policy_{host}, nonce
	Drone checks policy
③ **Drone → Host**:	Signed attestation quote
	Host checks attestation/freshness
④ **Host → Drone**:	Okay to enter

Figure 2: The check-in protocol.

3 REGULATING DRONE BEHAVIOUR

In this section, we describe how the vision discussed above can be realized on drones equipped with the ARM TrustZone. The methods used by the host to determine whether a drone is in policy compliance also depend on the software stack running on the drone. We describe a concrete implementation for drones running the Robot Operating System (ROS) [24]. We make the case for *dynamic information-flow control* (IFC) as the core policy enforcement mechanism within the drone's software stack, and describe how ROS lends itself well to IFC.

3.1 Background on the ARM TrustZone

The TrustZone is a set of security extensions to the ARM architecture. A TrustZone processor executes in one of two "worlds," a *secure world* or a *normal world*, with the transition between the two mediated by a security monitor. The normal world executes complex applications within a rich computing environment, and is the world with which the end-user interfaces during regular operation of the platform. The secure world consists of security-related applications, and implements functionality such as secure boot, which prevents bootup of devices in which there are unauthorized modifications to the software stack. Device memory can be partitioned between the two worlds such that a partition is reserved for exclusive access by the secure world. This allows the secure world to store sensitive information, such as cryptographic keys, that are not accessible to the normal world, which is untrusted.

3.2 The Check-in Protocol

As shown in Figures 1 and 2, check-in is a four-step protocol:

① The drone identifies itself with its public key Pub_{drone} and the corresponding public key certificate. We assume that the Pub_{drone} uniquely identifies the drone and that it is registered with a suitable regulatory authority. The host verifies the certificate with the regulatory authority.

② The host sends its privacy policy together with a random nonce to the guest drone. The drone determines whether the policy is acceptable to it. In the subsequent discussion, for instance, we show that these policies can be expressed as information-flow control restrictions. If not, it can abort the protocol and navigate away from the restricted space.

③ If the policy is acceptable, the drone sends a signed *attestation quote* [27] back to the host. The attestation quote contains a digitally-signed hash-chain of the software stack installed in the drone's normal world, and is produced by the secure world. The nonce from step ② is also included as part of the signed attestation. The quote allows the host to check that suitable policy-enforcement software is installed on the drone. Note that the host uses Pub_{drone} from step ① to verify the digital signature, thus tying the message received in step ③ to the drone from step ①. The presence of the nonce in the quote convinces the host that the response is fresh.

④ If all checks succeed, the host authorizes the drone to enter its restricted space.

The above protocol establishes to the host that the guest has policy-enforcement software installed on it. This software could itself execute in the untrusted normal world, with the secure world ensuring that the enforcement code cannot be disabled or modified at runtime. Several widely-deployed systems adopt this approach. For example, the Samsung Knox platform [5] uses this approach to protect against unauthorized modifications to key data structures of the normal world's operating system kernel. As discussed in the following sections, we adopt a similar approach, where the normal world tracks information-flow across applications and peripherals. The normal world contains the enforcement *mechanisms*, the code of which remains unchanged as the drone moves from one restricted space to another. Thus, the secure world simply needs to attest to the host that these mechanisms exist on the drone, and protect the normal world from unauthorized modifications. The *policy* to be enforced would vary based on the host, and the secure world places the policy to be enforced within normal world memory (from where the enforcement mechanism reads and enforces it) and write-protects it (see Figure 3).

3.3 Dynamic Information-flow Control

Dynamic information-flow control (IFC; also called taint-tracking) mechanisms regulate the flow of data objects within a system. IFC mechanisms work by attaching *taint labels* to data objects. The taint of an object changes as it is processed by various entities on the system. IFC employs flow rules that determine how labels can change and whether certain label transitions are allowed. When a data object reaches a *sink* (*e.g.,* an output channel), the label attached to the object can be used to determine whether it can be transmitted via the sink. IFC has successfully been applied in a number of domains, including most notably, to tracking data leaks from Android applications [12].

IFC is well-suited to our setting because it can be used to express a number of useful privacy policies. For example, the policies discussed in Section 1 can be expressed with IFC:

① The *Process-Images-Locally* policy can be enforced by tagging images obtained from the camera with a special label (*e.g.,* `camera`), and ensuring that data objects with the `camera` label cannot be stored on disk or accessed by the network interface.

② The *Blur-Exported-Images* policy can be enforced by ensuring that only a trusted blur-filter application, which executes in the normal world and is recorded as part of the attestation quote sent to the host, can change the label of a data object from `camera` to `blurred-img`. The network interface is allowed to transmit objects with a `blurred-img` label but not those with a `camera` label (see Figure 3).

③ The *Log-GPS* policy can be enforced by ensuring that all readings from the GPS sensor, *e.g.,* tagged `GPS`, pass through a trusted logging application. The logging application executes in the normal world and interfaces with the secure world to create a tamper-proof audit log. The logging application is authorized to change the label

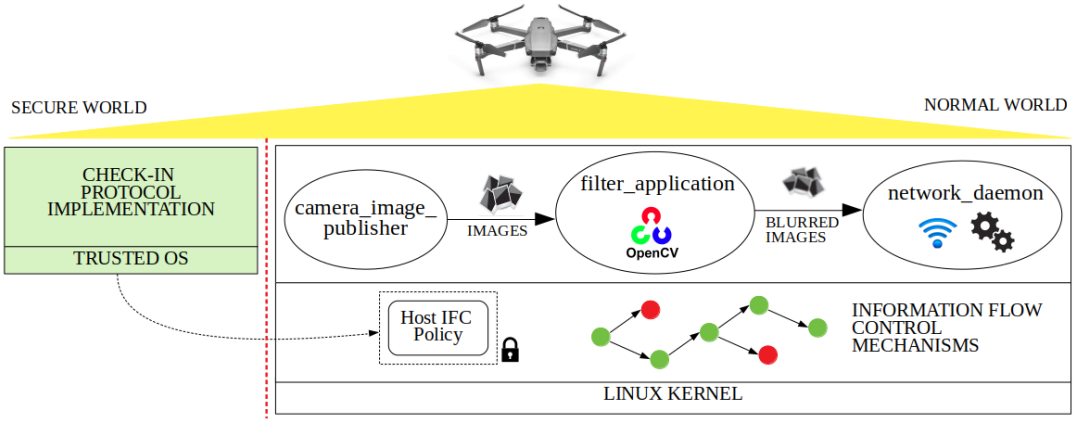

Figure 3: Policy enforcement within the guest drone. The normal world runs a version of ROS enhanced with IFC mechanisms. During check-in, the secure world attests to the host that the normal world runs an IFC-enforcing ROS version. The secure world places and write-protects the host's IFC policy in normal-world memory, and also protects ROS' security mechanisms from unauthorized modifications. In turn, ROS ensures that information-flow restrictions imposed by the host are followed. In this example, it ensures that only images that have been processed by a blur-filter can be read by the network interface.

of GPS sensor readings to `audited-GPS`, and other client applications on the drone (*e.g.,* control software) can only consume GPS readings with the tag `audited-GPS`.

Note that the normal world is only entrusted with the task of attaching labels to data objects and propagating these labels as the object is processed by various entities in the system. Label transitions could be implemented by trusted applications running atop the normal world, *e.g.,* the blur-filter and the logging application in the policies described above. These applications are *trusted by the host* in that the host requires the guest drone to execute them for policy compliance; it verifies the integrity of these applications using the attestation quote during check-in.

3.4 IFC on ROS

We built our IFC prototype on ROS. ROS is a set of middleware libraries that run atop Linux, and provides a convenient platform to author robotics applications. ROS is used by a number of drone manufacturers, *e.g.,* 3DR, Parrot, Gaitech, Erle, BitCraze, Skybotix, for developing robotics applications and supports a variety of hardware platforms. The ROS market, including drones and other kinds of robots, is forecasted to reach $400 million by 2026 [25]. A number of popular software packages for drones also build atop ROS. For example, FlytOS [1], which is an operating system for drones, and MAVROS [2], which is a communication driver for autopilots such as PX4 and ArduPilot that use the MAVLink communication protocol (a widely-used protocol for communicating with small drones), are built atop ROS.

ROS is a publish/subscribe system in which applications can either be publishers or subscribers (or both). Every publisher produces data objects on a certain *topic*. Subscribers can choose to listen to messages that fall under a set of topics. When applications start up on ROS, they register themselves as publishers or subscribers for a certain set of topics. ROS libraries then bootstrap communication by matchmaking publishers and subscribers based on the topics they advertised when they started up. It does so by setting up an IPC channel between the corresponding publishers and subscribers. All underlying communication then directly happens between the Linux processes that implement the publishers and subscribers. For example, the camera application could publish images under the topic `camera`, and all image-processing applications could choose to subscribe to this topic. ROS ensures that all messages associated with this topic (and only messages associated with this topic) are sent to all subscriber image-processing applications.

The architecture of ROS lends itself well to IFC. Because messages are already tagged with topics, we can directly leverage these topics as IFC labels. The IFC policy simply determines the topics of the data objects that subscribers can consume. Label transitions on data objects by trusted applications are implemented by registering the application as a subscriber for the topic of the input data object and a publisher for the output data object. For example, the trusted blur-filter application used to enforce the *Blur-Exported-Images* can register as a subscriber for the `camera` topic and publish objects with the topic `blurred-img`.

We were able to implement a rudimentary IFC mechanism atop ROS by adding just under 200 lines of Python code to the ROS codebase, and used it to enforce the example policies discussed above. Most of this code relates to parsing the host's policy and setting restrictions on the topics that applications can publish/subscribe. ROS's topic-matching algorithms in its publish-subscribe system then automatically take care of enforcement.

3.5 Challenges and Future Directions

We are continuing work to build an end-to-end IFC in drones. Our ROS-based implementation is only one part of the end-to-end system. While ROS bootstraps communication between publishers and subscribers, the underlying communication happens directly via sockets and IPC between Linux processes. An adversary could still bypass ROS-level IFC enforcement by directly setting up communication channels between Linux processes.

We are currently investigating IFC enforcement from within the normal-world OS to ensure that such attacks are not possible. In such a system, the kernel attaches labels to data objects managed

by the OS and propagates these labels as data is copied. We are currently building atop the IFC model proposed by Flume [18], which attaches labels to files, sockets, and other OS-level abstractions, and exposes a label interface to applications. We are adding support for IFC within the Linux kernel, and integrating ROS to run atop IFC-enabled Linux. With this support, applications running on the drone can specify their label requirements using a familiar interface, such as ROS-level topics. ROS communicates these to the OS, which manages and propagates the labels as the application executes. While this effort is a work in progress, our modifications to Linux and ROS largely mirror Weir [22], which implemented IFC atop Android using a Flume-like label model.

3.6 Deployment Considerations

The discussion thus far has assumed that a guest drone checks in with the host and obtains an IFC policy. A practical deployment must also consider the challenges involved in performing check-in. We discuss two possible deployment models:

① ***Designated entry corridors.*** A host may require that drones that wish to enter its restricted space do so via designated entry corridors. The host could then deploy physical infrastructure (*e.g.,* NFC or Wi-Fi access points) that records the identity of the guest drone and communicates the host's policy to the drone. Having a designated entry corridor also helps ensure that the drone has an area in which to wait and establish compliance with the host in case network connectivity is intermittent. This model is applicable in settings such as office and apartment complexes, defense establishments and in university campuses.

② ***Cooperation with wireless provider.*** A host could communicate the geographic coordinates of its restricted space to 5G or WiMax providers that service the area and also specify its IFC policy. A guest drone can then directly obtain this information from the 5G/WiMax provider. One of the risks with this approach is that if network connectivity is unreliable, then the drone may not reliably receive the host's policy prior to entry. A possible way to mitigate this risk is to pre-load a database of restricted spaces and the corresponding policies before the drone takes flight. This model is applicable for delivery drones where the set of locations to be visited is known *a priori.*

These models are ineffective for adversarial drones (or covertly deployed ones) that fly unauthorized over the host's restricted space. These cases currently fall outside of our threat model. Additional methods are needed to detect/prevent adversarial drones. Of course, even for the case where drones are used overtly, the techniques proposed in this paper are designed to benefit hosts rather than the owners/operators of drones. Lacking additional incentives or regulations, there is little motivation for drone manufacturers to incorporate the hardware and software stack modifications that we propose. However, given the increasing social and governmental concern over privacy, we hypothesize that such incentives and regulations are likely to emerge over time. For example, if a large commercial drone operator deploys these mechanisms, it may incentivize its competitors to also deploy them on their drones.

4 RELATED WORK

Restricted Spaces. The work most closely related to ours is that of Brasser *et al.* [6], which develops the notion of restricted spaces for ARM TrustZone-enabled mobile devices. While Brasser *et al.*'s work directly inspired this paper, there are a few key ways in which this paper deviates from that work. Brasser *et al.*'s work focused on host control over peripherals in guest devices, *e.g.,* to ensure that interfaces such as the camera, Wi-Fi, and Bluetooth on the guest device are used in accordance with the host's policies. Their work used host-initiated remote memory operations as the core mechanism for policy enforcement on guest devices. The ability of hosts to remotely modify memory operations results in a potential security backdoor via which malicious hosts may modify unsuspecting guest devices. Brasser *et al.* therefore introduced the notion of a vetting service that guest devices could use to verify that the host's memory updates do not maliciously modify the guest device.

In contrast to their work, this paper focuses on information-flow control between applications executing atop drones, and uses an enhanced ROS stack on the guest drone for policy enforcement. This change overcomes two key drawbacks in Brasser *et al.*'s approach. First, Brasser *et al.* employed peripheral control for policy enforcement, *i.e.,* a peripheral would simply be turned off as specified by the host's policies. We extend Brasser *et al.*'s approach beyond peripheral control to fine-grained control over data produced and consumed by peripherals. For example, autonomous drones rely critically on the use of their camera for navigation, and it would be infeasible to turn the camera off. In contrast, our finer-grained approach allows the camera to operate, but instead places restrictions on how the pictures/video captured by the camera are used.

Second, Brasser *et al.*'s work used remote memory operations on guest devices to inject policy-enforcement code. This approach is intrusive on the guest device and requires additional supporting infrastructure if the guest does not trust the host, *e.g.,* the vetting service mentioned above. In contrast, our approach builds the policy-enforcement mechanism directly into the drone (*e.g.,* atop ROS), and the host's restrictions are simply encoded as policies to be enforced by the drone. The host does not inject any new code into the drone and no vetting service is required.

Drone and UAV Security. Prior work on drone and unmanned aerial vehicle (UAV) security has largely focused on exploitable vulnerabilities in either the communication channels or the hardware/software stack on the drone. Such attacks have focused on exploiting unencrypted communication over wireless media to implement eavedropping, cross-layer attacks, signal jamming, denial of service, and dropping Wi-Fi communication with ground control, to name a few [14, 15, 23, 30, 31]. Other attacks on drones involve GPS spoofing attacks to fool the drone into moving to a different destination (possibly with the intention of hijacking the drone) [16, 29]. Such attacks are a concern for delivery drones, which may be captured and analyzed to obtain sensitive details related to delivery data [28].

More recent work has investigated how ARM TrustZone may be used to enhance drone security. PROTC [19] is a system that leverages the TrustZone to address the lack of system-level protection for a drone's peripherals. PROTC ensures that applications runnning on the drone are able to securely access its periperhals

even when the operating system of the drone has been compromised. Liu *et al.* [20] investigate the problem of ensuring a drone has not strayed into no-fly zones. To do this, they leverage the TrustZone to keep a tamper-proof log of the drone's GPS locations, which then provide an alibi to a third-party auditor that the drone was in compliance. However, they do not consider the problem of ensuring that the drone respects a host's privacy policies when it is within permitted airspace.

ROS Security. There has been a modest amount of research into identifying the key security vulnerabilities of ROS, and proposals by which these can be overcome [9]. McClean *et al.* [21] analyzed ROS and identified a number of vulnerabilities such as plain text communication between nodes, unprotected TCP ports, unencrypted data storage and a lack of an authentication mechanism for nodes.

Some of the key security vulnerabilities within ROS are inherent in the publish-subscribe paradigm that it employs. Publishers are unable to control the consumption of their data, and subscribers are unable to verify the integrity of received messages. Rodriguez *et al.* [26] have proposed to use message level encryption between nodes and find that it is a feasible solution even for low-power robots. Prior work has also proposed integration of the ROSC++ package with TLS to provide end-to-end encrypted communication channels [9]. Dieber *et al.* [10] develop an authentication mechanism that allows publishers and subscribers verify each others' identities and establish the integrity of messages exchanged between them. These and similar enhancements are under consideration as part of the Secure ROS [32, 33] framework, which is under active development.

5 SUMMARY

Drones are now widely available and are soon proposed for use in commercial settings. These drones will pose a massive threat to security and privacy unless active steps are taken now to develop suitable policy regulations and enforcement technologies. This paper takes a step in that direction by proposing the notion of restricted spaces for drones and demonstrating an IFC-based mechanism to enforce privacy policies on drones.

Acknowledgments. We thank the reviewers for their comments and Silvia Santini for shepherding the paper. This work was supported in part by a Ramanujan Fellowship from the Government of India and by the Robert Bosch Centre for Cyber-Physical Systems.

REFERENCES

[1] FlytOS: Operating system for drones. https://flytbase.com/flytos/.
[2] MAVROS – MAVLink extendable communication node for ROS with proxy for ground control station. http://wiki.ros.org/mavros.
[3] 112th Congress. FAA Modernization and Reform Act of 2012, February 2012. https://www.congress.gov/112/plaws/publ95/PLAW-112publ95.pdf.
[4] ARM. Security technology building a secure system using TrustZone technology (white paper). *ARM Limited*, 2009.
[5] A. Azab, P. Ning, J. Shah, Q. Chen, R. Bhutkar, G. Ganesh, J. Ma, and W. Shen. Hypervision across worlds: Real-time kernel protection from the ARM TrustZone secure world. In *ACM Conference on Computer and Communications Security*, 2014.
[6] F. Brasser, D. Kim, C. Liebchen, V. Ganapathy, L. Iftode, and A-R. Sadeghi. Regulating ARM TrustZone devices in restricted spaces. In *ACM International Conference on Mobile Systems, Applications, and Services*, 2016.
[7] A. Cavoukian. *Privacy and drones: Unmanned aerial vehicles.* Information and Privacy Commissioner of Ontario, Canada Ontario, 2012.
[8] Civil Aviations Authority (CAA). CAP 722, Unmanned Aircraft System Operations in UK Airspace - Guidance, March 2015. https://publicapps.caa.co.uk/docs/33/CAP%20722%20Sixth%20Edition%20March%202015.pdf.
[9] B. Dieber, B. Breiling, S. Taurer, S. Kacianka, S. Rass, and P. Schartner. Security for the Robot Operating System. *Robotics and Autonomous Systems*, 98, 2017.
[10] B. Dieber, S. Kacianka, S. Rass, and P. Schartner. Application-level security for ROS-based applications. In *2016 IEEE International Conference on Intelligent Robots and Systems*, 2016.
[11] Directorate General of Civil Aviation. Requirements for Operation of Civil Remotely Piloted Aircraft System (RPAs), August 2018. http://dgca.nic.in/cars/D3X-X1.pdf.
[12] W. Enck, P. Gilbert, B-C. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth. Taintdroid: An information-flow tracking system for Realtime privacy monitoring on smartphones. In *ACM/USENIX Symposium on Operating System Design and Implementation*, 2010.
[13] Federal Aviation Administration (FAA). Small Unmanned Aircraft Systems (sUAS), June 2016. https://www.faa.gov/documentLibrary/media/Advisory_Circular/AC_107-2.pdf.
[14] K. Hartmann and C. Steup. The vulnerability of UAVs to cyber attacks-an approach to the risk assessment. In *IEEE International Conference on Cyber Conflict*, 2013.
[15] A. Y. Javaid, W. Sun, V. K. Devabhaktuni, and M. Alam. Cyber security threat analysis and modeling of an unmanned aerial vehicle system. In *IEEE Conference on Technology for Homeland Security*, 2012.
[16] A. J. Kerns, D. P. Shepard, J. A. Bhatti, and T. E. Humphreys. Unmanned aircraft capture and control via GPS spoofing. *Journal of Field Robotics*, 31(4), 2014.
[17] C. Koettl and B. Marcolini. A closer look at the drone attack on Maduro in Venezuela, August 2018. https://www.nytimes.com/2018/08/10/world/americas/venezuela-video-analysis.html.
[18] M. Krohn, A. Yip, M. Brosdky, N. Cliffer, F. Kaashoek, E. Kohler, and R. Morris. Information flow control for standard os abstractions. In *ACM Symposium on Operating Systems Principles*, 2007.
[19] R. Liu and M. Srivastava. PROTC: Protecting drone's peripherals through ARM TrustZone. In *3rd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, 2017.
[20] T. Liu, A. Hojjati, A. Bates, and K. Nahrstedt. Alidrone: Enabling trustworthy proof-of-alibi for commercial drone compliance. In *IEEE International Conference on Distributed Computing Systems*, 2018.
[21] J. McClean, C. Stull, C. Farrar, and D. Mascareñas. A Preliminary Cyber-Physical Security Assessment of the Robot Operating System (ROS). In *Unmanned Systems Technology XV*, volume 8741, 2013.
[22] A. Nadkarni, B. Andow, W. Enck, and S. Jha. Practical DIFC enforcement on Android. In *USENIX Security Symposium*, 2017.
[23] J-S. Pleban, R. Band, and R. Creutzburg. Hacking and securing the AR. Drone 2.0 quadcopter: investigations for improving the security of a toy. In *Mobile Devices and Multimedia: Enabling Technologies, Algorithms, and Applications 2014*, volume 9030, 2014.
[24] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. ROS: An open-source Robot Operating System. In *ICRA workshop on open source software*, 2009.
[25] Transparency Market Research. Robot Operating System Market - Snaphshot, 2018. https://www.transparencymarketresearch.com/robot-operating-system-market.html.
[26] F. J. Rodríguez-Lera, V. Matellán-Olivera, J. Balsa-Comerón, Á-M. Guerrero-Higueras, and C. Fernández-Llamas. Message Encryption in Robot Operating System: Collateral Effects of Hardening Mobile Robots. *Frontiers in ICT*, 5, 2018.
[27] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a TCG-based integrity measurement architecture. In *USENIX Security Symposium*, 2004.
[28] S-H. Seo, J. Won, E. Bertino, Y. Kang, and D. Choi. A security framework for a drone delivery service. In *2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, 2016.
[29] D. P. Shepard, J. A. Bhatti, T. E. Humphreys, and A. A. Fansler. Evaluation of smart grid and civilian UAV vulnerability to GPS spoofing attacks. In *Radionavigation Laboratory Conference Proceedings*, 2012.
[30] E. Vattapparamban, İ. Güvenç, A. İ Yurekli, K. Akkaya, and S. Uluağaç. Drones for smart cities: Issues in cybersecurity, privacy, and public safety. In *2016 International Wireless Communications and Mobile computing Conference*, 2016.
[31] W. Wang, Y. Sun, H. Li, and Z. Han. Cross-layer attack and defense in cognitive radio networks. In *IEEE Global Communications Conference*, 2010.
[32] R. White, G. Caiazza, H. Christensen, and A. Cortesi. Using and developing secure ROS1 systems. In *Robot Operating System*, 2019.
[33] R. White, D. Christensen, I. Henrik, and D. Quigley. SROS: Securing ROS over the Wire, in the Graph, and through the Kernel. *arXiv preprint arXiv:1611.07060*, 2016.
[34] A. Young. Passenger jet carrying 240 people nearly hits a drone at 15,000ft, 2018. https://www.dailymail.co.uk/news/article-6172229/Passenger-jet-carrying-240-people-nearly-hits-drone-15-000ft.html.