A Hypervisor-Based Privacy Agent for Mobile and IoT Systems

Neil Klingensmith
University of Wisconsin
Computer Engineering
naklingensmi@wisc.edu

Younghyun Kim
University of Wisconsin
Computer Engineering
younghyun.kim@wisc.edu

Suman Banerjee
University of Wisconsin
Computer Science
suman@cs.wisc.edu

ABSTRACT

We present a design for a mobile and IoT data privacy agent that lives in software on end devices. Our privacy agent learns and enforces a user's privacy policy across all devices that he manages. Implemented as a hypervisor onboard the end device, our privacy agent sits between the device's hardware and its application software. It can inspect, modify, block, and inject I/O traffic between the device's main CPU and its peripherals. The key advantage of our architecture is that, unlike network middleboxes, the hypervisor can track all I/O transactions in *unencrypted* form. This makes our privacy agent potentially much more effective than those that only monitor network traffic because it can track and modify plaintext data. Our privacy agent also gives users the ability to impose a uniform privacy policy across all devices that they manage, which minimizes the burden and possibility of error that arise when setting privacy policy on individual devices. Since the notion of per-user (as opposed to per-app) privacy policy is relatively new, there has not been much opportunity for researchers to think about how to define and implement policy on that scale. We propose a method for learning a user's privacy policy one time and automatically implementing it in a context-aware fashion on multiple devices.

CCS CONCEPTS

• **Security and privacy → Embedded systems security**; *Mobile platform security*; • **Computer systems organization → Embedded software**.

KEYWORDS

Privacy; Mobile Systems; IoT; Hypervisors; Real-time

ACM Reference Format:
Neil Klingensmith, Younghyun Kim, and Suman Banerjee. 2019. A Hypervisor-Based Privacy Agent for Mobile and IoT Systems. 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/10.1145/3301293.3302356

1 INTRODUCTION

Data from IoT devices is particularly vulnerable to intrusion because it cannot be curated by users in the same way as a social media feed. Furthermore, because IoT devices often operate autonomously, we may forget that they are transacting in troves of our personal data, much of which is of a far more intimate nature than the cat pictures and status updates we share on social networks. This problem is compounded by the fact that users are increasingly interacting with multiple mobile and IoT devices. When, as many predict, we are routinely interacting with hundreds of autonomous devices, the problem of managing the privacy settings for each will be intractable.

In this work, we examine the possibility of deploying a verifiably independent privacy agent onboard IoT and mobile devices. It would implement a user's personal privacy policy across all IoT devices that the user manages. This privacy agent, under exclusive control of the user, can act as an intermediary between the device software and its cloud services platform. Our agent can inspect the flows and prune or modify potentially sensitive information before it can be released to the putatively untrusted cloud.

For users, the benefit of having a privacy agent is that they can assert control over the way their data is used. Instead of having to blanketly agree to the service providers' privacy policy for every IoT and mobile app, users would have the option of setting their own privacy policy for their devices.

We propose building a privacy agent that is implemented as a hypervisor onboard mobile and IoT devices. It lives between the system's hardware and its stock app, inspecting, modifying, and dropping data that is transmitted between the app and its peripheral hardware devices. Because these peripheral buses typically transmit data in unencrypted form, our privacy agent has access to data collected in plain text. Since it can see raw data, our privacy agent has the opportunity to implement targeted privacy policies that would not be possible with a network middlebox, which generally would only have access to encrypted data.

If a user has several devices outfitted with our hypervisor-based privacy agent, he can set his privacy policy once and allow all devices to implement it in a context-aware fashion. This would significantly alleviate the difficulty of deploying and managing many IoT devices, particularly if they are all different. However, without the user setting specific policy for each device individually, the system must infer those settings from a high-level description of how devices should behave in general. To our knowledge, this problem of inferring specific policy from a short description has not been investigated before. We discuss some ways of implementing this later in the paper.

For any intermediate privacy agent, there is a tradeoff between ease of deployment and effectiveness that is affected by how closely coupled it is to the IoT device itself. Existing work on the topic of intermediate IoT privacy policy has tended toward the more loosely coupled end of the spectrum because it has been the only approach with a reasonable hope of widespread adoption.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
HotMobile '19, February 27–28, 2019, Santa Cruz, CA, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6273-3/19/02…$15.00
https://doi.org/10.1145/3301293.3302356

Cinch [1] is another system that used a hypervisor to intercept I/O traffic for security, not privacy. Cinch intercepts traffic on the USB bus and redirects it through a virtual network port so standard networking security measures can be used to validate I/O traffic. Cinch is mostly about validating I/O traffic to prevent attacks to the relatively vulnerable USB driver stack. It does not consider the possibility of selectively anonymizing I/O data which requires (1) that we make different assumptions about the data and (2) that we dedicate significant computational resources to anonymization.

Security vs. Privacy. In security, we generally assume that there are good guys—the users—and bad guys who are trying to disrupt service or steal (computational resources, data, bandwidth, etc.) from the users. In privacy, the delineation is not so clear. The assumption of privacy is that users supply data to service providers, usually of their own volition and in exchange for access to some service. But the service provider may be overly zealous in collecting information about users. In privacy, there are not bad guys—there are only information gluttons. Unlike security, the goal of privacy protections should not be to cut off the flow of information from users to service providers but to limit the flow in a way that acceptable to both user and service provider.

The reason that this seemingly subtle difference matters is that in privacy measures, if we do not like the way a service behaves, we cannot necessarily just shut it down as we would in a security countermeasure. Usually we need to find a solution to allow the service to continue to operate in some capacity. In this work we present a practical mechanism for selective data anonymization which allows the service to continue operating, but it gives users control over what and how much data they share about themselves.

One important technical challenge is to intercept and modify a datastream nonintrusively and with low latency.

2 ARCHITECTURE

Our approach is to run IoT end device software inside a specialized hypervisor. The hypervisor will run on the end IoT device itself. It will emulate the bare-metal hardware interface provided by the specific end device so that the software will not know that it is running inside a hypervisor. A diagram is shown in Figure 1. Hermes is an IoT-optimized hypervisor that could be used for this purpose.

The IoT privacy agent is either a guest that runs inside the hypervisor or an external cloud-based service. When it receives I/O traffic from the hardware, the hypervisor passes the raw data directly to the privacy agent for anonymization according to the user's privacy policy. If the agent is implemented in the cloud, the hypervisor would stream the I/O traffic over a UDP connection, allowing for more sophisticated anonymization software at the expense of latency[1]. The privacy agent can anonymize the data in four general ways before returning it to the hypervisor to be passed to the IoT app:

(1) **Passthrough:** communication between app and peripheral proceeds normally without intervention from the hypervisor. The hypervisor may record salient data to decide if it is necessary to take a different action.
(2) **Drop:** hypervisor drops the communication between app and peripheral. For example, the hypervisor blacks out the image from a video camera.
(3) **Modify:** Hypervisor allows data exchange between peripheral and app, but modifies it in transit. For example, the user's location as reported by a GPS receiver may be modified to make the app think that the user is in a different location.
(4) **Inject:** hypervisor creates fake traffic that appears to be coming from the peripherals to the app. For example, the hypervisor on a smart watch may create fake I/O events that appear to be coming from the accelerometer to spoof the activity tracker.

The privacy agent can observe all I/O traffic exchanged between the IoT device software and its peripherals. As traffic flows back and forth to the peripherals, data can be inspected or modified by the privacy agent's peripheral data processing module. The privacy agent's mode decision engine selects modes for the data parsing module to operate in: passthrough, modify, drop, or inject. The privacy agent's user interface module provides a mechanism to allow the users to define policies that will be implemented by the mode decision engine and peripheral data processing module.

Others have proposed implementing privacy mediators inside network middleboxes that would inspect (and potentially modify) network traffic as it traveled between IoT devices and their respective cloud services [2, 3]. Our approach has two key advantages. First, it will not break if network traffic is encrypted (which it almost surely will be). Second, the hypervisor can control not just the IoT device's interface to the cloud but also its interface to its own hardware. This means that if the user wishes, he could black out a video stream before it can even be processed by the IoT device software or anonymize an audio stream to remove personally identifying information [7].

Because Hermes, unlike IoT device software, is open-source, users can evaluate for themselves whether they think it is secure enough to entrust with their data. But if the IoT device manufacturer cannot be trusted to safeguard the user's data, how can we be certain that it does not modify the hypervisor or bypass its data processing functionality?

One option is to use trusted platform modules (TPMs), which can perform software integrity attestation. A remote machine that lives either on an edge device or in the cloud could interrogate the Hermes-enabled IoT device, using a TPM to validate that the hypervisor is intact. Another option is to have users program their own devices with the hypervisor, similar to rooting an Android phone. This would remove the possibility of IoT device manufacturers disabling the hypervisor.

2.1 Hermes

Hermes [6] is a hypervisor for MMU-less microcontrollers that enables high-performance bare metal applications to coexist with RTOSes and other less time-critical software on a single CPU. Hermes is a single monolithic interrupt service routine that intercepts all CPU exceptions before they can be processed by the operating system. Since it is a relatively small piece of code, it should be possible to verify that it is secure. On boot, the Hermes initialization code sets up the CPU's exception table to point to the Hermes ISR.

[1] We have demonstrated this technique in the lab for video traffic with a few tens of milliseconds of additonal latency and a low frame loss rate. We will not present the results of these experiments here.

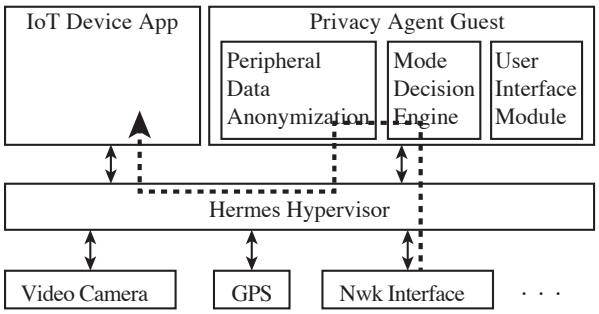

Figure 1: A diagram of the Hermes hypervisor, including our privacy agent, implemented as a guest running in parallel with the IoT device app. The dotted line shows that path traversed by I/O traffic from the hardware to the device's app.

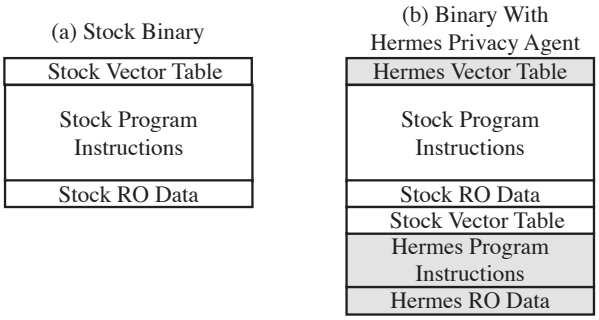

Figure 2: A diagram of the IoT device's software binary, before and after adding the Hermes hypervisor.

It then launches the guest operating systems in the ARM CPU's unprivileged execution mode. In benchmark tests, we have found that I/O responsiveness is far more deterministic under Hermes than under FreeRTOS, the most popular open-source real-time OS. Depending on the type of I/O event, guests running inside Hermes may experience higher I/O latency than apps running on the bare metal (without an operating system or hypervisor), but the increase is comparable to server and workstation virtualization platforms like VMWare and Xen.

2.2 Dynamic Taint Analysis

We propose using dynamic taint analysis similar to Panorama and TaintDroid [4, 9] to track how data flows from I/O peripherals to network connections or storage on IoT devices. Using the hypervisor's ability to inject data into the stock IoT or mobile app, we can insert a taint source at the hardware level and track the tainted data to a destination of interest.

Taint propagation, the process by which we track tainted data as it moves through memory buffers, relies on the hypervisor's ability to trap when a tainted memory region is accessed. Memory protection units (MPUs), available on higher-end ARM Cortex-M CPUs, would allow us to generate an exception when the IoT or mobile app accesses a tainted buffer.

MPUs are like extremely simple versions of memory management units. Their only function is to generate a memory protection exception when a process accesses a memory region that it does not own. Region permissions can be configured by the hypervisor to cause an exception any time the stock IoT or mobile app accesses a tainted buffer. The exception could then be handled by the hypervisor to record the access.

Using dynamic instruction analysis, the hypervisor would be able to generate a taint graph that shows how tainted data propagates from source to sink. This would be an additional tool that could be used to determine whether an app's behavior is consistent with the user's privacy policy.

2.3 Deploying a Virtualized Privacy Agent

Since the Hermes privacy agent lives in the IoT device's main memory, it must be built by someone who is technically proficient enough to reflash an IoT device. For average users, installation be as easy as rooting a phone if the device has appropriate firmware loading capabilities. The process of building the rooted firmware image is the more technically challenging piece, and that is what we discuss here.

Because of the way that the ARM Cortex-M CPU handles some privilege violations, the IoT software's kernel may need to be patched to be compatible with Hermes. In the ARM Cortex-M core, some privileged instructions, when executed in user mode, will not cause an exception and will instead complete as a `NOP` instruction. For example, the `mrs` and `msr` instructions[2] are classified as privileged instructions, but when they are executed in unprivileged mode, they fail silently: the register write is not committed, and the processor continues normal execution.

The problem is that if a guest OS tries to modify the processor state with one of these privileged instructions, that state modification cannot be registered by Hermes since it does not cause an exception. The privileged instruction will complete like a `nop` instruction without modifying the CPU state. Critical CPU state changes like disabling interrupts will not work as intended.

This can cause some important changes to the CPU state to go unregistered (see reference [6] for details). As a workaround, we must patch the OS kernel, adding `undefined` instructions directly following the privileged instructions in order to be sure the hypervisor has an opportunity to trap the privileged instructions and emulate them. When the hypervisor encounters an undefined instruction exception, it will search backward in the instruction stream for an `mrs` or `msr` instruction and emulate it. If we run an unpatched kernel inside the hypervisor, it will crash because the intended CPU state modifications will not happen as intended.

To do this, we would need to disassemble the stock binary and insert `undefined` instructions following the problematic ones. This would also require some branch targets and read-only data references to be relocated.

On the ease-of-deployment spectrum, our solution tends toward the more tightly-coupled end, making it more difficult to deploy than a middlebox approach. To ensure wide-scale adoption, it would be best for our privacy agent to be flashed into firmware by the

[2] `mrs` and `msr` are most frequently used to enable/disable interrupts and modify stack pointers in an RTOS.

device manufacturer, either during factory programming or a field update.

However, like rooting an Android phone, it would be possible for users to reflash their own IoT and mobile devices. We envision the process of user programming the Hermes-based privacy agent to work as follows:

(1) Read the contents of the IoT device's main program memory into a binary file. This contains the device software's interrupt vector table, program instructions, and read-only data (Figure 2 (a)).
(2) Copy the stock vector table to a different location in the binary image. Replace it with the Hermes vector table[3]. Append the Hermes program instructions and read-only data to the binary file (Figure 2 (b)).
(3) Modify the program's instruction stream, inserting `undefined` instructions after privileged instructions to be emulated.
(4) Reflash new binary image to the IoT device.

Steps 1 and 4 may require specialized hardware to access the device's program memory. Other authors [5] have shown that it is possible to directly read and write an IoT device's main memory with no special hardware on some devices. Naturally, the process of reading and writing memory would be highly device-specific. Steps 2 and 3 could be carried out automatically by a script.

2.4 Driver Implementation

We have built three preliminary implementations of our privacy agent, one in the Linux kernel, one in FreeRTOS (an embedded OS for microcontrollers), and one in Hermes. Our goal was to understand how driver and hardware interfacing will affect the scalability of our proposed system. In all implementations we built a simple data path in which the mode decision engine and data anonymization blocks are stubs that just pass data through to the device app. Our demo app is an IoT video camera in all cases.

In the Linux kernel we found the process of modifying the USB video driver to be very time consuming and tedious, largely because the driver code lacks comments and documentation. Also, Linux's USB video driver is tightly coupled to video4linux, a logically separate kernel module that presents a video interface to userspace apps. We think that adding privacy agent functionality for other device types would be similarly complex and probably not scalable. Also, implementing the privacy agent in the OS requires the end device to run Linux, which may not be the case for all devices.

The issues in the FreeRTOS implementation were similar to those in Linux with the additional requirement that we needed the complete source code (application plus OS) in order to add our privacy agent. This is because microcontroller software is compiled into a single binary, so we cannot modify the OS while keeping the userland code in place.

The implementation in Hermes was much more seamless, mainly because we did not need to modify any existing drivers. Hermes sends video data to the privacy agent VM in raw form. The privacy agent can modify the data as needed and send it to the IoT device app without worrying about the particulars of how the device app manages internal data structures. The processor's data sheet, typically freely available, specifies the peripheral's data format, which is what the IoT device app expects to receive from the peripheral. As long as the privacy agent VM generates data in the appropriate format, things will work correctly. Also, since Hermes is a virtualization environment, it presents the same interface to the IoT device runtime software as the hardware. This allows us to add the privacy agent as a separate module while leaving the IoT device software untouched.

[3]The application's original (stock) vector table is needed to initialize it as a guest inside the Hermes hypervisor.

	Frame Loss Rate	Frame Jitter (σ)
FreeRTOS, Low I/O Load	0 fps	7.03 ms
FreeRTOS, High I/O Load	4.2 fps	47.91 ms
Hermes Low I/O Load	0 fps	4.85 ms
Hermes High I/O Load	0 fps	4.86 ms

Table 1: Performance comparison of Hermes and FreeRTOS in the video app. Low frame loss rate and jitter is better.

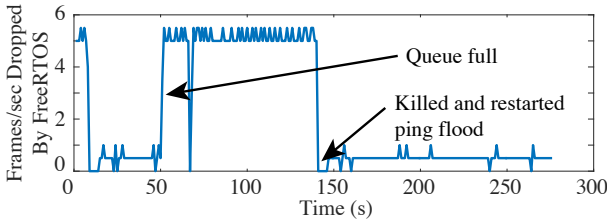

Figure 3: Number of frames dropped by FreeRTOS per second as a function of time while camera was running in the presence of a ping flood.

2.4.1 Evaluation. In the FreeRTOS and Hermes implementations, we measured frame jitter and loss rate under varying I/O load conditions. Both were benchmarked on an Atmel SAME70 Xplained development board with an ARM Cortex-M7 CPU. The board has a VGA image sensor interface that can accept images in 16-bit RGB color format at a rate of about 10 frames per second. We present an evaluation both as a proof of concept for our privacy agent and as a demonstration that it can be used without diminishing performance (in most cases).

In both cases, we saw no frame losses under low I/O load. Figure 3 shows the number of frames dropped every second during a 276-second test of the FreeRTOS-based privacy agent. Note in the figure that a queue overflow in the RTOS caused a high rate of frame drops under high I/O load. This will not happen in the Hermes implementation because it can be configured to prioritize certain kinds of I/O traffic over others. **Under high I/O load, Hermes lost no frames, and the jitter did not increase** because the hypervisor prioritizes userland video frame processing over lower-priority networking interrupts. Table 1 shows a performance comparison between Hermes and FreeRTOS for the video app under low and high I/O load conditions. Figure 4 shows histograms of the inter-frame timing for the same test cases.

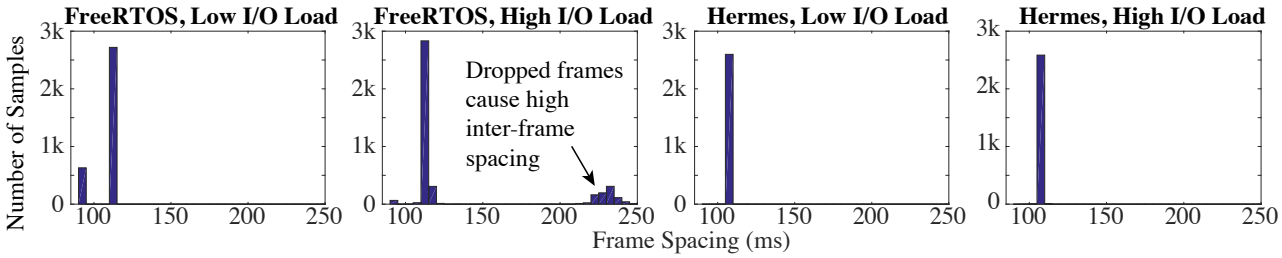

Figure 4: Histograms of inter-frame spacing for our privacy agent, implemented in different runtime environments. We call the standard deviation of these histograms the *jitter* in Table 1.

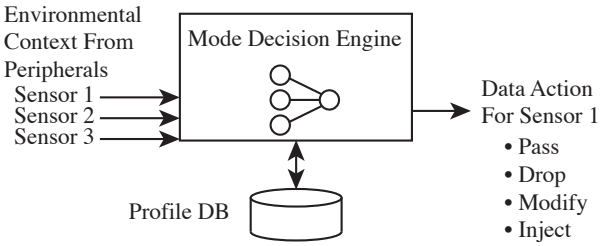

Figure 5: A diagram of the mode decision engine, which uses the privacy policy to decide what action to take with data collected from peripheral devices.

2.5 Privacy Policies

To make our system seamless, we need a way for users to quickly express their privacy policy, similar to [8]. The user's policy should be implemented across all the IoT devices under their control. A privacy policy should begin with a short (one paragraph or less) human-readable explanation of how a user wants his data managed. It should apply generally to all devices under a user's control. In the course of daily device use, the system may occasionally ask the user questions to clarify the policy. Answers will be stored locally and shared with the user's other devices. A neural network in the mode decision engine will make decisions about how data should be anonymized based on the user's responses to privacy policy questions. Its inputs will be (1) data collected from local peripheral devices and (2) the user's responses to targeted questions about data consumption.

2.5.1 Expressing a Privacy Policy. During setup, users are asked to choose a privacy profile that describes, in broad terms, their preferences regarding how the privacy agent will handle their data. Example profiles are:

- **Laissez Faire:** generally permit apps to do as they please with the user's data. Only flag data usage that may be in violation of an app's privacy policy, and alert the user in those cases.
- **Camera-shy:** track video and scrutinize its content. Optionally black out or blur the faces in video frames containing certain people.
- **Secret agent:** don't disclose user's identity (name, email, etc.) or GPS location to cloud services.

The user-selected profile serves as a starting point for the privacy agent. Running on the mobile device, it will take data collected from peripherals and pass it through the profile mode decision engine, which will decide how to handle it (see Figure 5). The mode decision engine will generate one of the four actions outlined above: passthrough, drop, modify, or inject. When the mode decision engine's neural network produces a low-confidence classification, it will query the user to clarify what the privacy policy should be in that situation. User queries will help the mode decision engine refine a personalized privacy policy from the coarse profile.

2.5.2 Implementing Privacy Policies. Some data will have a clear action associated with it: for example, in the camera-shy policy, frames with no faces or people should be passed through without modification. But some will fall in a gray area, like video frames with faces that the policy engine has not seen before. During the course of their device use, users will be asked targeted questions about whether to permit certain kinds of data collection by their mobile and IoT devices in order to clarify the user's policy. Answers to these questions will be stored in the profile database to inform future data handling decisions. The profile database's input-output pairs can be used to train the mode decision engine's classification algorithm. When the database is updated, the updates are pushed to the privacy agents on all devices under the user's control. The mode decision engines on each end device can then be retrained using the new database.

3 APPLICATIONS

Security Camera. suppose a user buys a security camera with cloud-based analytics that can identify suspicious activity within the home. The platform vendor (Nest, Belkin, etc.) advertises that the device comes equipped with the Hermes privacy agent, allowing the user to set his own data privacy policy for the device. This is the first Hermes-enabled IoT device that this user has installed, so he will need to configure his privacy policy for the Hermes privacy agent.

After installation, the user connects the device to the home WiFi network. He then visits a third-party website that validates the presence and integrity of the Hermes privacy agent using the a trusted platform module onboard the camera. This website also asks him to set his privacy policy profile, as outlined in Section 2.5.1. The user chooses both Secret Agent and Camera Shy profiles, and the privacy agent setup is complete.

The Hermes peripheral data parsing modules (Figure 1) monitor video frames as they pass from the camera hardware to the IoT device firmware. The data parsing module passes each frame through the mode decision engine, which interprets the user's privacy policy and decides what action to take (passthrough, drop, modify, or inject). Suppose that most frames contain nothing of interest, so the mode decision engine passes them unmodified to the IoT app, which sends them on to the device manufacturer's cloud-based analytics engine.

Then, the user wanders in front of the camera. The mode decision engine detects his face and notices that it is not in its profile database. It passes the frame up to the IoT app, but it also sends the user an email asking whether or not he wishes frames with his face to be passed, blurred, or blocked in the future. The user requests to have all frames with his face blocked in the future. From then on, the mode decision engine sends black frames to the IoT app software when it detects the user's face in the frame.

Smart Watch. suppose the user subsequently buys a Hermes privacy agent-enabled smart watch. He will again go through the device-specific setup process and the third-party Hermes integrity validation. This time, though, he will not need to configure his privacy profile—the existing profile database will be pushed from the camera to the watch after the integrity validation completes, and he can continue using his new smart watch without any further setup.

As the privacy agent captures I/O transactions and processes them through the mode decision engine, it notices that the user's coordinates are being continuously uploaded to the smart watch vendor's cloud-based server. It identifies this activity as a violation of the secret agent privacy profile, which does not permit transmission of the user's personally-identifying information. The privacy agent begins blocking all I/O transactions to and from the smart watch's GPS receiver and sends an email to the user to inform him that the GPS is being blocked, giving him the option to unblock it if he chooses.

When he receives this second email about GPS data collection by his smart watch, the user decides that the features enabled by GPS tracking are valuable enough to allow it. He enables GPS coordinate collection by the smart watch via the privacy agent's user interface. This decision is sent to the smart watch, which updates its profile database and pushes the updates to the other Hermes privacy agent-enabled devices in the user's home, including the security camera. Once the devices have received their updated profile databases, they retrain their mode decision engines with the new information.

4 CONCLUSION

In this work, we proposed building a mobile and IoT data privacy agent within an embedded hypervisor. The agent would sit between the device's data collection peripherals and the stock IoT software, modifying or pruning data in order to implement the user's individual privacy policy. Instead of asking users to set privacy policies individually for each IoT device, the privacy agent could learn the user's privacy preferences contextually by observing how the user interacts with devices and asking targeted questions about those interactions. A learned privacy policy would be used to configure new devices. This would make IoT device deployment much easier by reducing the burden of configuring detailed privacy policies for every device. It would also give the user considerably more control in dictating his own privacy policy because the privacy agent would have the power to disable or modify data streams that the user is not comfortable sharing with the service provider.

The techniques we discussed in this work provide a platform to deploy data anonymization software on mobile and IoT devices without cooperation from device manufacturers or service providers. Data anonymization algorithms themselves are relatively new, and their capabilities and limitations are not well understood. In order for our techniques to be useful (and trustworthy), it will be important to think more about information theoretic limits of anonymization algorithms and how those limitations would affect the data transacted by our privacy agent.

5 ACKNOWLEDGMENTS

We would like to acknowledge the anonymous reviewers and our shepherd Dr. Eric Rozner for his detailed comments and guidance. The authors were supported in part by the US National Science Foundation through grants CNS-1719336, CNS-1647152, CNS-1629833, and CNS-1343363.

REFERENCES

[1] Sebastian Angel, Riad S. Wahby, Max Howald, Joshua B. Leners, Michael Spilo, Zhen Sun, Andrew J. Blumberg, and Michael Walfish. 2016. Defending against Malicious Peripherals with Cinch. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX Association, Austin, TX, 397–414. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/angel

[2] Andy Crabtree, Tom Lodge, James Colley, Chris Greenhalgh, Kevin Glover, Hamed Haddadi, Yousef Amar, Richard Mortier, Qi Li, John Moore, Liang Wang, Poonam Yadav, Jianxin Zhao, Anthony Brown, Lachlan Urquhart, and Derek McAuley. 2018. Building accountability into the Internet of Things: the IoT Databox model. *Journal of Reliable Intelligent Environments* 4, 1 (01 Apr 2018), 39–55. https://doi.org/10.1007/s40860-018-0054-5

[3] Nigel Davies, Nina Taft, Mahadev Satyanarayanan, Sarah Clinch, and Brandon Amos. 2016. Privacy Mediators: Helping IoT Cross the Chasm. In *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications (HotMobile '16)*. ACM, New York, NY, USA, 39–44. https://doi.org/10.1145/2873587.2873600

[4] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. 2010. TaintDroid: An Information-flow Tracking System for Realtime Privacy Monitoring on Smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation (OSDI'10)*. USENIX Association, Berkeley, CA, USA, 393–407. http://dl.acm.org/citation.cfm?id=1924943.1924971

[5] Grant Hernandez, Orlando Arias, Daniel Buentello, and Yier Jin. 2014. Smart nest thermostat: A smart spy in your home. *Black Hat USA* (2014).

[6] Neil Klingensmith and Suman Banerjee. 2018. Hermes: A Real Time Hypervisor for Mobile and IoT Systems. In *Proceedings of the 19th International Workshop on Mobile Computing Systems & Applications (HotMobile '18)*. ACM, New York, NY, USA, 101–106. https://doi.org/10.1145/3177102.3177103

[7] Anantharaghavan Sridhar, Neil Klingensmith, and Suman Banerjee. 2016. dB-Hound: Privacy Sensitive Acoustic Perception in Home Settings: Poster Abstract. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems CD-ROM (SenSys '16)*. ACM, New York, NY, USA, 370–371. https://doi.org/10.1145/2994551.2996711

[8] Primal Wijesekera, Joel Reardon, Irwin Reyes, Lynn Tsai, Jung-Wei Chen, Nathan Good, David Wagner, Konstantin Beznosov, and Serge Egelman. 2018. Contextualizing Privacy Decisions for Better Prediction (and Protection). In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, New York, NY, USA, Article 268, 13 pages. https://doi.org/10.1145/3173574.3173842

[9] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. 2007. Panorama: Capturing System-wide Information Flow for Malware Detection and Analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*. ACM, New York, NY, USA, 116–127. https://doi.org/10.1145/1315245.1315261