Verifying Internet of Things Safety and Security in Physical Spaces

Z. Berkay Celik | Purdue University
Patrick McDaniel and Gang Tan | Pennsylvania State University
Leonardo Babun and A. Selcuk Uluagac | Florida International University

Concerns about safety and security have led to questions about the risk of embracing the Internet of Things (IoT). We consider the needs and techniques for verifying the correct operation of IoT devices and environments within the physical spaces they inhabit.

The growth of Internet of Things (IoT) devices that integrate online processes and services with the physical world has had profound effects on society. From smart homes to personal monitoring devices and manufacturing automation, IoT applications have changed the way we live, work, and play. In fact, my smartwatch interrupted my writing this paragraph with a reminder; such interactions are examples of the rapidly changing way in which smart devices pervade our daily lives. Yet, while users and industry alike have broadly embraced IoT systems, we have yet to understand the implications of these devices on our safety and security.

Incidents threatening user safety and security have caused concern about the risks of embracing IoT-augmented lives and led to fervent calls to restrict the technology's use. These risks are far from merely academic: vulnerable and faulty devices can lead to everything from compromised baby monitors[1] to vehicle crashes and monetary theft.[2] In other domains, failures could cause serious health consequences in the form of compromised IoT pacemakers[3] and even result in catastrophic environmental damage from pipeline explosions.[4]

Much like traditional security problems, many of these failures are a consequence of software bugs, user error, poor configuration, and faulty design. Some of the other causes represent new classes of failures: interactions within the physical spaces that lead to unsafe or insecure environments. For example, devices might have conflicting goals: An IoT door lock may try to lock the door to secure the house, while a smoke alarm wants to keep residents safe by unlocking the door during an emergency. In these latter failures, individual devices might be operating correctly but jointly create a dangerous environment.

So, what do we do about this? What we need is some way to model the interactions between devices and verify not just one device but the joint behavior of all of the devices in the environment. In the sections that follow, we describe an approach that uses software verification through model checking to perform this analysis.

Architecture of IoT Systems

IoT systems integrate physical processes with digital connectivity. While several IoT platforms have emerged

Digital Object Identifier 10.1109/MSEC.2019.2911511
Date of publication: 10 June 2019

 Copublished by the IEEE Computer and Reliability Societies

in various domains, they follow a common structure providing a software stack to monitor and control IoT devices. Figure 1 shows the components in a typical IoT platform: IoT devices, a hub, a cloud back end, and IoT applications (apps). In addition, some IoT platforms integrate with external services and allow user interaction through mobile apps.

IoT devices are equipped with embedded sensors and actuators. Sensors detect properties or changes in the physical world and generate events to notify IoT applications, while actuators are the actions that a device can perform. For example, a door may have "opening," "opened," "closing," and "closed" sensor readings but only "open" and "close" actuators. The hub controls communication between IoT devices and the cloud back end. The communication is established through network protocols implemented inside the hub. These protocols are selected depending on requirements such as low power or the need for a lossless connection. The cloud back end creates software proxies that act as a conduit for physical devices. It also runs IoT applications and provides services for the remote control and monitoring of devices.

Among the IoT's most attractive features is support for custom automation in the form of apps. For example, an IoT app in a smart home might unlock a door when its presence sensor notifies it that a user has arrived and lock the door once the user is in the house. IoT apps are event-driven; they subscribe to device states or other predefined events, such as mobile app interactions. An event handler is invoked to handle each event, which may lead to further events and actions. IoT apps may send or acquire information from external services through the Internet. For instance, an app may connect to a weather forecasting service and send its location information to the service to get the local weather and set the room temperature value. IoT platforms often provide users with a proprietary mobile app that is used to add and configure devices and to install IoT apps from a market. Apps are usually vetted prior to publishing, requiring the developer to submit source code.

IoT Safety and Security

Although users and industry have widely embraced the IoT, concerns have emerged about the safety and security of physically and digitally augmented lives.[5–8] Safety and security issues may result from misuse of IoT devices by an individual IoT app. For instance, a faulty app may unlock the front door of a house when the user is not at home or create undesired conditions by turning off the heat in cold weather.[5] Since modern IoT devices are often embedded within an environment of many apps and services, apps may cause security violations when they interact with other apps, the physical world, and digital services. These interactions pose unique challenges to the automatic discovery of safety and security violations.

Multiple-App Interactions

When multiple apps are colocated in a shared environment, the interactions among the devices that the apps control can lead to undesired device states. Here, the final environmental state does not depend on an individual app; it is the result of multiple interacting apps. Figure 2 illustrates examples of interactions among IoT devices and digital services. In general, IoT apps may interact in several ways.[5,9]

1. An app's event handler might change device attributes and trigger events in another app; for example,

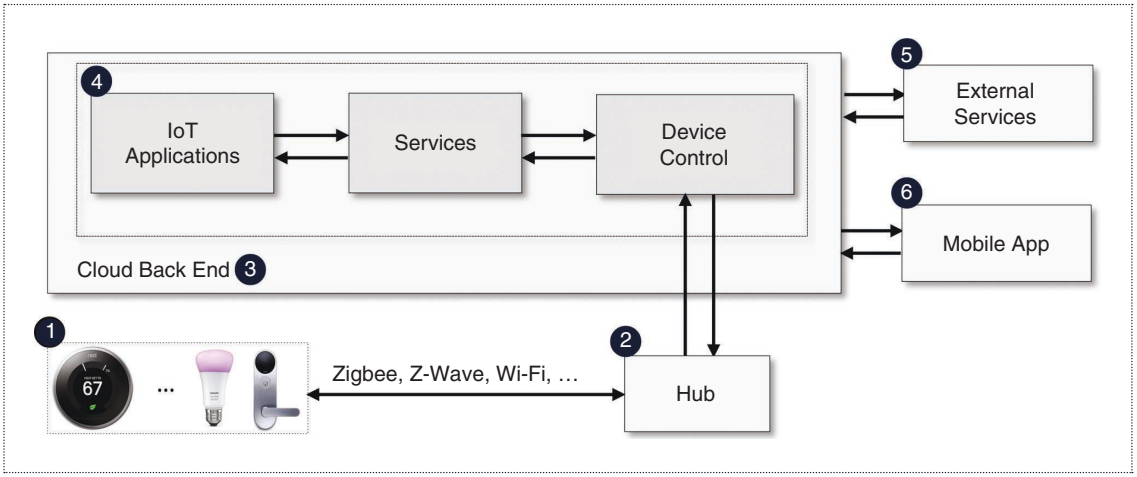

Figure 1. An example architecture of an IoT system.

an app switches on the living room light when there is smoke, and another app locks the front door when the light is switched on, potentially exposing a user to risk from a fire [Figure 2(a)].
2. Multiple apps can change the same attribute of a device; for example, an app turns off the alarm when a user is present at home, while a motion-detected event turns on the alarm, leading to a potential race condition [Figure 2(b)].
3. Apps could respond to the same event in conflicting ways; for example, when a contact sensor is open, one app switches the light on, while another app switches the light off [Figure 2(c)].

Additionally, apps may interact through mode attributes, which are behavioral filters to automate device actions. For instance, an app that changes the "home" mode to the "away" mode when a user leaves home interacts with an app that uses the "mode change" event to unlock the door [Figure 2(d)].

Interactions Between Apps and the Physical World

Device operations enable IoT apps to indirectly interact with each other through shared physical environments, such as air, temperature, and humidity, therein leading to unsafe/undesirable states.[7,10] Here, an app may change the physical environment through a device action, which subsequently causes other apps to react in unexpected ways. For instance, an app may turn on the heater to raise the temperature of a house and, when the temperature exceeds a threshold, another app might open the windows [Figure 2(e)]. This could allow a burglar to break into a home through a window by controlling the house's temperature.

Interactions Between Apps and the Digital Domain

The interactions in an IoT environment are broader than the devices and the physical space and extend into the digital domain via trigger-action platforms.[9,11] Trigger-action platforms, such as If This, Then That (IFTTT) and Microsoft Flow, enable users to write apps that connect IoT devices to digital services. These apps allow users to set an event in a service to trigger the desired action in another service automatically. For example, a trigger-action app turns on the light when the user receives an email, and another app logs the user's presence to a public log when the front door is unlocked [Figure 2(f)]. This intertangled environment represents real risks. For instance, integrity violations result from an untrusted action changing a trusted attribute (an untrusted email turns on the light or unlocks a door), and confidentiality violations occur when an action changes an attribute that makes the private information publicly available (when an "unlock" function leaks a door state or user location to a public log).

We observe that the interaction problem in the IoT is similar to the feature-interaction phenomenon in telecommunications in the early 1980s. Feature interactions arise when features (functions or services) are used together; they modify or influence each other in a way that leads to inadvertent interactions and produces undesired side effects.[12] Consider the interaction between the call forwarding and call-waiting features of a telephony system. When both features are active, the system can reach a possibly unsafe state when it receives a call on a busy line—the system has no specific response about suspending or forwarding the call. It is unquestionable that the IoT is here and that the deployment of new devices and services will increase rapidly, yet feature interactions and the resulting side effects in IoT environments have not been fully explored.

Toward IoT Code Verification

There is hope here; we can use program verification to identify not only flaws and vulnerabilities in implementations but also to model the physical world's state

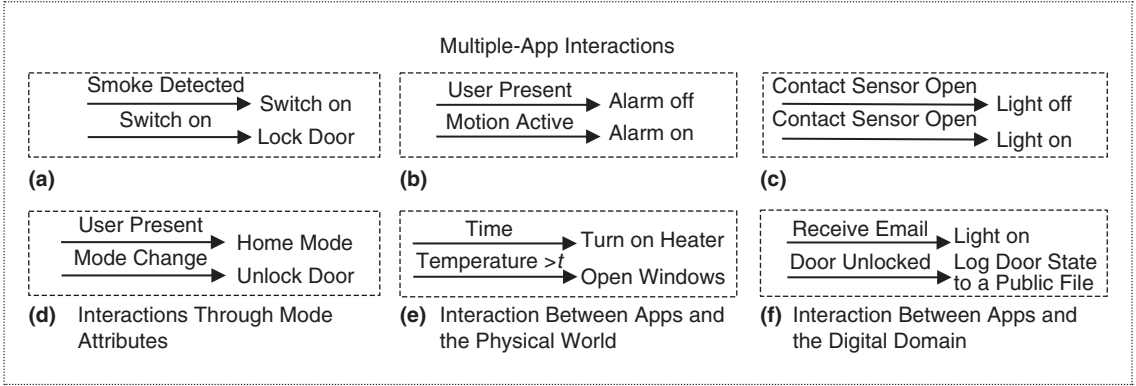

Figure 2. An illustration of (a)–(f) various kinds of interactions among IoT devices and digital services. *t*: room temperature value.

transitions and their potential to harm users or environments. Program verification is often used to analyze the correctness of software in safety-critical systems against some property.[13] We refer to the property as a system artifact that can be expressed in a specification and validated formally using a validation technique. Here, we focus on model checking, where systems are represented as state models (for example, transition systems), and these models are checked against specified properties using a model checker, such as NuSMV.[14] This either confirms that the properties hold on the model or produces counterexamples when the properties do not hold. The challenge is extracting state models and finding a systematic way of identifying the properties that are appropriate for a given IoT and deployment domain.

The central insight that allows us to make progress in this exceptionally difficult domain is that IoT development platforms are highly structured, allowing tractable analysis of complex properties.[10,15] These platforms almost universally follow the sensor-computation-actuator design paradigm. Sensors observe physical processes and trigger events. These events, in turn, trigger app event handlers. In processing events, apps actuate devices, which may also trigger further events. Such structures enable us to accurately model an IoT app as a state model, which maps device attributes to states and events to transitions that are readily analyzable using model checking.

We present two example IoT apps, smoke alarm and water-leak detector, and use their state models to illustrate interactions among devices. Figure 3(a) presents their events and actions. The smoke-alarm app controls a smoke-detection alarm and a water valve. The app triggers the alarm and opens the water valve when smoke is detected; it also activates fire sprinklers when a certain heat level is reached. The water-leak detector app detects a water leak by using a moisture sensor and closes off the main water-supply valve to prevent any further damage. Figure 3(b) illustrates the state models of the smoke alarm and water-leak detector apps extracted from their source code. For instance, the state model of the water-leak detector app starts from an initial state S_0 and transits to state S_1 when a leak is detected, and the state transitions are controlled by the output of the leak detector: "water leak-detected" (leak).

Another key technical question here is what properties should be used to check the state models. Of course, the effectiveness of device analysis is determined by the quality of properties being checked. The challenge is knowing what kinds of properties are desirable to use to verify the IoT environments. Intuitively, we can check properties stating that an IoT device's state should transition according to its specified functionality. For example, a smoke alarm should always sound by enabling the alarm actuator when the smoke sensor indicates smoke, regardless of the device configuration, implementation details, and other environmental factors. More broadly, we can check device 1) safety, which states that the device will not harm the user or environment, 2) security, which is the property ensuring that a device's function or data cannot be subverted by an adversary, and 3) functionality, which states that the device will behave according to a known specification. Here, interactions among devices may naturally cause safety, security, and functionality issues.

We have studied use/misuse case-requirements engineering to identify IoT safety properties in targeted domains.[5] Here, we focus on practical IoT domains by structuring properties on transitions of physical/digital IoT environments. This approach derives requirements (properties) by assessing the connections between 1) assets, which are artifacts that

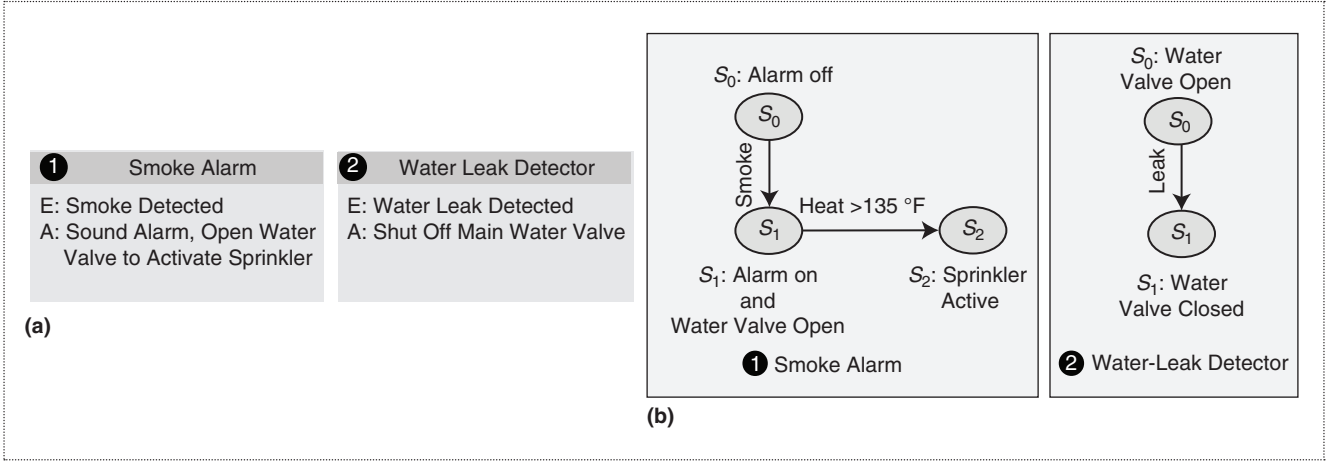

Figure 3. An example of IoT implementations and their state models. (a) Events (E) and actions (A) of example IoT implementations. (b) A state model of IoT implementations.

someone places value on, for instance, a garage door; 2) functional requirements, which define how a system is supposed to operate in a normal environment; for example, when a garage-door button is pressed, the door opens; and 3) functional constraints, which restrict the use and operation of assets. For example, a garage door must only open when an authorized device requests it.

We present example properties for the apps depicted in Figure 3. We verify a safety property on the state model of the smoke-alarm app; for example, "Does the alarm always sound when there is smoke?" To perform this analysis, we encode the safety property in temporal logic and verify it on the state model with a model checker. The analysis shows that the smoke alarm works as expected; it sounds the alarm when smoke is detected. To check properties when both the water-leak detector and smoke-alarm apps are colocated in an environment, we build a state model from the union of multiple apps' state models. The resulting state model represents the composite behavior of the apps running together. The unified state model then is checked against properties. Here, we verify the model against the property, "Does the sprinkler system activate when there is a fire?" The analysis reveals that there was a safety violation: The water-leak detector app shut off the water valve and stopped the fire sprinklers when it detected them releasing water. In this case, the joint behavior of the otherwise safe devices leaves users at risk from a fire.

Real-World IoT Code Verification

For our initial foray into this space, we developed two proof-of-concept frameworks for IoT analysis. Soteria[5] is a static-analysis system that extracts state models from IoT apps' source code and validates whether an IoT app or IoT environment adheres to identified properties through model checking. Yet, static analysis has limitations in overapproximating IoT states and state transitions, leading to false positives. For instance, the analysis may extract an imprecise model that indicates that the door may be unlocked when the user is not at home, while the original source code does not have this behavior. To overcome this issue, we developed IoTGuard,[9] a dynamic-analysis system that enforces identified properties by monitoring the device-execution behavior at runtime. Being dynamic, IoTGuard more precisely tracks IoT states and state transitions using runtime information, and it can deal with new devices dynamically plugged into an IoT environment. IoTGuard responds to property violations either by blocking property-violating device actions or by asking users to approve or deny violations through runtime prompts.

We used the frameworks to study a data set of 95 IoT and trigger-action apps, including SmartThings official and community-contributed IoT apps and IFTTT trigger-action official apps. SmartThings supports the most devices among smart-home platforms, and IFTTT is a widely employed trigger-action platform with millions of users and apps. The SmartThings apps in our data set control 20 different IoT devices, including smart lights, energy meters, door locks, smoke detectors, and water-leak detectors. The IFTTT apps in our data set connect IoT devices with 12 different applications, including email, Twitter, and Google services. The apps provide device functionality in categories such as safety and security, green living, convenience, home automation, and personal care. We developed 37 safety and security properties for smart-home platforms. To illustrate, one general property stated that an event handler must not change a device attribute to conflicting values, such as opening and closing a door at the same time. Another identified device-specific property stated that the door must always be locked when the user is not at home, thus ensuring the safe use of the door. Through these studies, we evaluated property violations caused by the IoT and trigger-action apps, both in isolation and when colocated in an environment.

Our analysis uncovered 10 individual apps violating 11 properties. Further, our study of the composite behavior of 25 interacting apps uncovered 16 property violations that were not found in the isolated apps. Table 1 summarizes the analysis results on individual and multiple apps. For instance, four apps interact with each other. First, an app changes the location mode to "away" or "sleeping" when the switch is turned off. Changing the location further triggers three apps, leading to unauthorized control of heater and air conditioner temperature values. Furthermore, an app turns off a set of devices, including a smoke detector and a security system, while another app turns on a set of appliances, such as a coffee machine, slow cooker, and heater when the user is not at home or is sleeping.

These studies demonstrated that many apps violate properties when used in isolation and together in multiple-app environments. We plan to expand our analyses to support more platforms as well as study more complex interactions between users and IoT environments.

What Now?

How do we move forward? Our experience suggests that the IoT developer community should extend current validation and testing practices. Before allowing a new device to enter the market, it must be evaluated not only for correctness in isolation but

Table 1. An example of property violations in individual IoT apps and IoT environments.

Analysis	Number of apps	Number of property violations	Example property violations
Individual apps	10	11	The flood sensor sounds the alarm when there is no water.
			The lights turn on and off when nobody is at home.
			The door is unlocked at sunrise and locked at sunset.
			The music player is turned on when the user is sleeping.
			The light switch is turned on when a hashtag used in social media and a missed call are received.
Interacting apps	25	16	The air conditioner and heater are turned on at the same time.
			The security system is turned off when the user is sleeping.
			The coffee machine, slow cooker, TV, and heater are turned on when the user is not at home.
			The window blinds and shades are opened when the doorbell is pressed.

also in environments of diverse IoT devices and configurations. This effort should seek to address the certification of composable IoT systems. Academic, industry, and government efforts need to integrate analysis techniques and systems designs to certify IoT devices and apps with respect to relevant properties. Such certification should be equivalent to a National Information Assurance Partnership Common Criteria Evaluation and Validation program for the IoT, in which regulations would systematically identify properties for specific IoT devices, frameworks, and environments and taxonomize IoT property classes. Because these regulations would require property-compliant IoT implementations and help vendors and customers assess risks, they could have a potential impact on user and environmental safety and security. However, such a change introduces several key challenges for academia. We next discuss a few research directions to encourage contributions from the community.

The size and complexity of the state space of IoT implementations may prevent easy analysis for most nontrivial properties. First, IoT apps receiving events from many devices can lead to state spaces with too many states and transitions, an oft-encountered issue in software verification that makes verification intractable. Second, IoT systems, such as vehicle control systems, might have a large amount of code to analyze to extract state models. To extract more compact models, we need more investigation into techniques, such as compartmentalization, that partition a large IoT app into smaller components.

If we can identify the core components that users are interested in, the model extraction can focus on those components, which would enable the extraction of smaller state models. For example, suppose an app controls a smoke sensor and an alarm; if the logic for the two devices is independent, and the user is interested in only the behavior of the smoke sensor, we can perform partitioning to get the component just for the smoke sensor. On the other hand, if the alarm's behavior may affect the smoke sensor's behavior, the system should be able to model the alarm's and sensor's dependency and deduce the behavior of both devices. To perform property checking on large code bases, code validation via techniques that do not require constructing state models, such as fuzzing, could be extended to generate sets of events and inputs to detect property violations. This would provide scalable checking at the expense of full verification.

Deciding what properties to verify systematically is crucial for an IoT domain. While we have extended a requirements-engineering process to identify IoT properties, it requires a certain level of domain expertise and human interaction. This can be a problem in highly complex IoT environments, where incorrectly identified properties can lead to falsely blocking legitimate states and failing to identify unsafe and insecure states. To address these issues, techniques related to safety and security-property discovery, including security-quality requirements egineering and the comprehensive, lightweight application-security process, as well as industrial methods, including Oracle's Software Security Assurance and Microsoft's Security Development

Lifecycle, can be explored. Other approaches would be to adapt machine learning and other modeling techniques to automate the property-discovery process in IoT devices and domains, which would entail profiling the events and actions of apps to construct models from which properties will be derived.

Finally, we should plan for response policies. Complex systems such as these are naturally going to have property violations. Response policies dictate the right course of action to take when these violations occur. Simply blocking a device state upon a property violation or asking a user for approval via runtime prompts could be dangerous. To keep an IoT environment stable when a property violation occurs, we must develop several response strategies that consider the severity of the problem and aim to preserve system integrity.

The IoT has reached critical mass, and the deployment of new devices and services will only continue to increase. We, as a computing community, need to manage this transition in ways that prevent accidents and malicious misuse of these new environments. In the end, and much like what we have learned about the Internet itself, we need to consider security and safety not only in terms of individual devices but as environments of digitally and physically interacting systems. ■

References

1. O. Waxman, "Stranger hacks into baby monitor and screams at child," *Time*. Accessed on: Feb. 15, 2019. [Online]. Available: http://time.com/79170/stranger-hacks-into-baby-monitor-and-screams-at-child/
2. G. Veerendra, "Hacking Internet of Things (IoT): A case study on DTH vulnerabilities," SecPod Technologies, Bangalore, India, White Paper, 2016. [Online]. Available: https://www.secpod.com/resource/whitepapers/Hacking-IoT-A-Case-Study-on-Tata-Sky-DTH-Vulnerabilities.pdf
3. H. Taylor, "How the Internet of Things could be fatal," CNBC. Accessed on: Feb. 15, 2019. [Online]. Available: https://www.cnbc.com/2016/03/04/how-the-internet-of-things-could-be-fatal.html
4. A. Jablokow, "How the IoT helps keep oil and gas pipelines safe," PTC. Accessed on Feb. 15, 2019. [Online]. Available: https://www.ptc.com/en/product-lifecycle-report/how-the-iot-helps-keep-oil-and-gas-pipelines-safe
5. Z. Berkay Celik, P. McDaniel, and G. Tan, "Soteria: Automated IoT safety and security analysis," in *Proc. USENIX Annu. Technical Conf. (USENIX ATC)*, 2018, pp. 147–158.
6. E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. 2016 IEEE Symp. Security and Privacy (SP)*, pp. 636–654.
7. W. Ding and H. Hu, "On the safety of IoT device physical interaction control," in *Proc. 2018 ACM Computer and Communications Security (CCS)*, pp. 832–846. [Online]. https://www.sigsac.org/ccs/CCS2018
8. D. T. Nguyen, Z. Q. Chengyu Song, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel, "IoTSAN: Fortifying the safety of IoT systems," in *Proc. Int. Conf. Emerging Networking Experiments and Technologies (CoNEXT)*, 2018, pp. 191–203.
9. Z. Berkay Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. Network and Distributed System Security Symp. (NDSS)*, 2019. doi: 10.14722/ndss.2019.23326.
10. Z. Berkay Celik, E. Fernandes, E. Pauley, G. Tan, and P. McDaniel, "Program analysis of commodity IoT applications for security and privacy: Challenges and opportunities," *ACM Comput. Surveys*, 2019. doi: 10.1145/3333501.
11. M. Surbatovich, J. Aljuraidan, L. Bauer, A. Das, and L. Jia, "Some recipes can do more than spoil your appetite: Analyzing the security and privacy risks of IFTTT recipes," in *Proc. 26th Int. World Wide Web Conf.*, 2017, pp. 1501–1510.
12. P. Zave, "Feature interactions and formal specifications in telecommunications," *IEEE Comput.*, vol. 26, no. 8, pp. 20–28, 1993. doi: 10.1109/2.223539.
13. E. M. Clarke Jr, O. Grumberg, D. Kroening, D. Peled, and H. Veith, *Model Checking*, 2nd ed. Cambridge, MA: MIT Press, 2018.
14. A. Cimatti et al., "NuSMV 2: An open source tool for symbolic model checking," in *Proc. Int. Conf. Computer Aided Verification*, 2002, pp. 359–364.
15. Z. Berkay Celik et al., "Sensitive information tracking in commodity IoT," in *Proc. 27th USENIX Security Symp.*, 2018, pp. 1687–1704.

Z. Berkay Celik (celik.berkay@gmail.com) is an assistant professor in the Computer Science Department at Purdue University, West Lafayette, Indiana. His research investigates the design and evaluation of security for software and systems, specifically focusing on emerging computing platforms and the complex environments in which they operate. Celik received a Ph.D. in computer science and engineering from Pennsylvania State University, State College.

Patrick McDaniel (mcdaniel@cse.psu.edu) is the William L. Weiss Professor of Information and Communications Technology in the School of Electrical Engineering and Computer Science at Pennsylvania State University (Penn State), State College. McDaniel received a Ph.D. in computer science from the University of Michigan, Ann Arbor. He a Fellow of both the IEEE and the Association for Computing Machinery, New York. He is also the director of

Penn State's Institute for Networking and Security Research, which focuses on the study of networking and security in computing environments.

Gang Tan (gtan@psu.edu) is the James F. Will Career Development Associate Professor in the Computer Science and Engineering Department at Pennsylvania State University (Penn State), State College. His research interests are software security, programming languages, and formal methods. Tan received a Ph.D. in computer science from Princeton University, New Jersey. He leads Penn State's Security of Software Group, which is broadly interested in applying programming language and compiler techniques to improving computer security.

Leonardo Babun (lbabu002@fiu.edu) is a graduate student and CyberCorps Scholarship for Service fellow in the Department of Electrical and Computer Engineering at Florida International University, Miami. His research interests are cyberphysical systems and Internet of Things security and privacy. Babun received an M.Sc. in electrical engineering from Florida International University, Miami.

A. Selcuk Uluagac (suluagac@fiu.edu) is an associate professor in the Department of Electrical and Computer Engineering at Florida International University, Miami, where he leads the Cyber-Physical Systems Security Lab. His research focuses on security and privacy for the Internet of Things and cyberphysical systems, and he has many publications on the practical and applied aspects of these areas. Uluagac received a Ph.D. in electrical and computer engineering from the Georgia Institute of Technology, Atlanta.

Digital Object Identifier 10.1109/MSEC.2019.2933708