Research Statement

Amit Levy

My research interests center on the design and implementation of secure systems. My work is guided by two principles: (1) system security should not impede third-party developers, who are often the main source of innovation, and (2) systems that secure third-party extensions also improve security by reducing the amount of specially-privileged *first-party* code.

Today, very few systems adhere to these principles. For example, Facebook's web platform exposes a limited subset of user data to applications due to security and privacy concerns. But many features require greater access to data, and hence can only be implemented by privileged Facebook engineers. This restricts innovation in features such as the news feed to a tiny fraction of developers. It also leaves user data more vulnerable when privileged code is buggy.

This problem isn't just the result of poor system building. It is hard to design highly extensible systems that are both secure and useful. Moreover, I believe that security and extensibility *must* be evaluated end-to-end, under real-world usage by actual practitioners. Most of the systems I have designed and built evolved through four phases of research:

1. **Identifying the problem** of systems that fail to realize their potential by handicapping third-party developers.
2. **Seizing opportunity** to attract a meaningful user base to a clean-slate system.
3. **Engineering** a system that addresses previous limitations while maintaining or improving security.
4. **Evaluation** of the new system. Because I frequently pursue the twin goals of security and extensibility, my systems cannot be evaluated in a vacuum. They must be evaluated *empirically*, by observing how they hold up to real third-party developers. Evaluation may involve promoting adoption of open source implementations, engaging with developers to foster a community, or even hiring inexperienced developers to use a research platform. Invariably, it also involves feeding lessons from the user base back into the system design.

I've used this approach to build systems for the Internet of Things, web platforms, and low-power wireless networking. I've published the results of this research in top conferences (including SOSP, OSDI, and MobiSys), released open-source implementations, and built a community of researchers and practitioners around one of them.

1 Tock: Internet of Things

Low-power microcontrollers are increasingly prevalent in the Internet of Things. These devices have extreme memory constraints—typically 16-512 kB of RAM. They also lack hardware features, such as virtual memory, that are integral to the design of modern operating systems. These constraints preclude traditional isolation abstractions, such as processes or microkernel services, leading to systems in which every line of code is fully trusted.

The result is that device manufacturers cannot safely run third-party applications on their microcontroller systems. Worse, these siloed devices end up incorporating large quantities of third-party code *anyway*, in the form of open source libraries or device drivers. A bug in any of this code makes the whole system vulnerable to attackers. If the Internet of Things continues to grow as predicted, we risk ending up with vulnerable and siloed devices running the world around us.

Fortunately, there's a huge opportunity here. Despite their widespread use, microcontrollers have no entrenched operating system and typically run non-portable applications. Hence, compatibility, the perennial barrier to research OS adoption, largely does not apply. Practitioners can easily adopt a new embedded operating system if it strengthens security, improves driver robustness, and enables harnessing of third-party application developers.

At the same time, other advances have opened new design possibilities for a clean-slate embedded OS. The success of Rust means that, for the first time, we have a robust developer community behind a language simultaneously providing memory safety and

complete control over memory allocation, deallocation, and layout. New hardware, such as the ARM MPU, offers non-traditional hardware-based memory protection. Between the compiler and hardware, these are powerful mechanisms for isolating untrusted code even in memory-constrained systems, but they cannot easily be retrofitted to legacy systems.

To take advantage of this opportunity, I developed Tock [13], a new operating system for microcontrollers. Through novel isolation abstractions, Tock makes it safe for end-users to run untrusted third-party applications and protects the kernel from buggy drivers. Tock isolates the memory and performance of applications using a preemptive process-like abstraction enforced in hardware by the MPU. The kernel is written in Rust and provides a type-safe API for building kernel components that ensures isolation of memory faults at virtually no runtime cost.

Tock's problem space presented unique technical challenges. With limited memory, any global heap allocation is a serious threat to system stability. On the other hand, the kernel cannot anticipate applications' resource demands ahead of time. Tock resolves this tension with a new memory management mechanism called "grants." Grants manage process-specific kernel heaps, ensuring the needs of one process do not affect the capabilities of another [8].

We originally built Tock for our specific hardware and use cases, but, frankly, it's not that surprising that Ph.D. students could run their own applications on their own operating systems. The *real* test was whether other developers could write applications for Tock and how the security properties would hold. To answer this question, we encouraged adoption and sought feedback by leading tutorials for both novice and experienced embedded developers [14–16]. We also introduced Tock in the classroom to teach embedded programming [4], wrote documentation, and engaged online with both companies and hobbyists.

The result of this effort is a growing community of Tock developers from both academia and industry. Academic researchers are using Tock to build a city-scale sensor network [11]. Helium (a well-funded ~50 person start-up in the Internet of Things space) is using Tock to build the next version of their programmable Internet of Things module [7]. Google ported it to their hardware root-of-trust [12] for use in some internal security products [10].

Fostering an ecosystem around Tock provided more than just validation of the architecture, it provided unexpected results. For example, we thought that process isolation would, at best, not hamper developer productivity. We learned that many developers actually preferred using processes, e.g., because the operating system could provide useful crash reports. On the other hand, community experience revealed that asynchronous I/O interfaces are error prone for developers who don't understand kernel internals, particularly when low-level operations are hidden in library code and used in more complex applications. As we continue answering questions like how to specify policies, how to account for power consumption, and how to efficiently configure resources, we've gained the ability to draw on applications and workloads from an active community of practitioners instead of guessing, *a priori*, what will work best with applications.

2 Hails: Web Platforms

A small number of large web sites, such as Facebook, curate most user data online. On the surface, these sites are becoming software *platforms* by giving third-party applications access to some data via APIs. In practice, though, most data access is reserved for the platform itself since end-users have to trust third-party applications completely with whatever data the applications can access.

The web platform restricts *which* applications can access data, but not *how* applications use the data. For example, if Facebook allows a third-party application to access a user's photos, it's up to that application to make sure the photos are not displayed to other, unauthorized users. Is it possible to ensure end-user security policies are enforced end-to-end without restricting third-party applications' capabilities?

One opportunity is a shift in how developers build and deploy web applications. Many developers are now deploying web applications on Platforms-as-a-Service (PaaS), like Heroku, Amazon Beanstalk, and Google AppEngine. These platforms alleviate the need for developers to manage their own servers. In exchange, de-

velopers adhere to the platform's programming conventions, such as choice of language, and build applications to work with databases and other services available on the platform.

Perhaps the deployment platform can enforce security policies on user data end-to-end? Instead of restricting which applications can access data, the trusted platform can track data across applications. This matches the policies users actually care about: who can see their data, not which developer's code can manipulate or access it along the way.

Hails [5] is a framework specifically designed to address the developer barriers erected by sites such as Facebook. In Hails, third-party applications run on the platform's servers instead of servers run by the developer. The trusted platform ensures that applications that have seen sensitive data can't communicate with users, files, databases, etc., that are not authorized to see that data. However, all functionality is implemented by the applications themselves. For example, we created a GitHub-like code sharing platform built entirely of untrusted applications.

Of course, the ultimate goal is to make sure ordinary developers can build secure platforms and applications using Hails, not just domain experts (i.e., me and the other Hails authors). To evaluate this, I supervised four undergraduates in their first year of programming who built a web platform and applications over the course of a summer internship. Another group of more experienced students did the same during a summer internship at MIT Lincoln Labs. In both cases, the students successfully built interesting applications, but found security policies difficult to express. This experience led us to create a DSL for writing policies and change how policies interact with the database.

Some of my collaborators founded a company, Intrinsic [6], that is porting Hails to Node.js.

3 Beetle: Wireless Protocols

Mobile phones, wireless routers, and other gateway devices allow wireless peripherals to communicate with local and cloud applications using specialized networking protocols like Bluetooth Low Energy. Unfortunately, intuitively simple use cases, such as logging heart rate in one application while viewing it in another, are impossible. Current operating systems simply do not allow applications to share access to Bluetooth Low Energy peripherals safely, resulting in barriers for developers.

I found an opportunity to address this deficiency in GATT, the application layer protocol Bluetooth Low Energy devices already use to communicate with applications. GATT is an ideal protocol for multiplexing access to devices. It has an attribute data-model. Endpoints perform operations such as `GET`, `WRITE`, and `NOTIFY` on the attributes. This is enough information for the operating system to let applications safely share access to peripherals without explicit coordination, and without changing the peripherals or the applications.

I designed and built Beetle [9], a new hardware interface for Bluetooth Low Energy. Beetle interposes on the GATT protocol between applications and devices. The system can distinguish between, say, a command to discover peripheral capabilities or fetch a reading from one that unsubscribes from future sensor reading updates. As a result, Beetle allows shared access to peripherals from multiple applications, fine-grained access control to peripheral resources, and transparent access to peripherals over a network.

Because GATT is not specific to any particular class of peripheral devices, Beetle works with existing peripherals and applications without requiring the operating system to understand any specific peripheral's functionality. For example, I ran Beetle on my own Android phone for several months so I could use two fitness apps that concurrently connect to a heart-rate monitor.

4 Industrial Impact

During my PhD I co-founded MemCachier, the largest provider of memcache to PaaS web applications. The opportunity for MemCachier was born of the same observation as Hails: that developers had begun deploying web applications on platforms like Heroku. These applications require additional services, like a distributed cache, that the platform does not provide and that can be done much more cost-effectively for a large number of small applications using a multi-tenant system.

MemCachier, like many other commercial products, is the result of low-hanging fruit (and the material need to pay rent in the Bay Area) rather than cutting edge technical advances. Nonetheless, deploying such a system at scale has resulted in benefits to the research community, as data from MemCachier traces enabled several projects by other researchers [1–3].

5 Future Work

Most academic research does not directly result in deployed systems, nor should it. However, my experience with Tock has taught me how advantageous deploying systems from an academic setting can be when trying to effect a radical change across an industry. Large companies like Google, small companies like Helium, academic sensor network researchers, and open source communities may not typically collaborate on a greenfield project. Nonetheless, they are all investing resources in Tock.

My work on Tock raises a variety of research questions about how to support multiprogramming on constrained platforms. These include how to prevent individual applications from exhausting the system's power budget; how to configure the system for optimal power consumption given a dynamic, untrusted application workload; and how to specify and enforce end-user security policies.

I will continue to look into other opportunities for secure and extensible systems. For example, "serverless computing" (a model of cloud computing where developers schedule granular functions rather than virtual machine instances) opens a space to simultaneously give applications more access to hardware features and schedule resources more efficiently.

Academia plays a critical role in shaping the software that future generations of developers will build. Academic research expands the frontier of what developers imagine they can build. By centering system security while striving to maximize developer potential, we can help create computing that is more usable, more accessible to developers, and safer.

References

[1] Blankstein, A., Sen, S. and Freedman, M.J. Hyperbolic Caching: Flexible Caching for Web Applications. *USENIX Annual Technical Conference* (July 2016).

[2] Cidon, A., Eisenman, A., Alizadeh, M. and Katti, S. Cliffhanger: Scaling Performance Cliffs in Web Memory Caches. *13th USENIX Symposium on Networked Systems Design and Implementation* (March 2016).

[3] Cidon, A., Rushton, D., Rumble, S.M. and Stutsman, R. Memshare: a Dynamic Multi-tenant Key-value Cache. *USENIX Annual Technical Conference* (July 2017).

[4] EE 285/CS 241: Embedded Systems Workshop: *http://web.stanford.edu/class/cs241/*.

[5] Giffin, D.B., Levy, A., Stefan, D., Terei, D., Mazières, D., Mitchell, J.C. and Russo, A. Hails: Protecting Data Privacy in Untrusted Web Applications. *10th USENIX Symposium on Operating System Design and Implementation (OSDI)* (October 2012).

[6] Intrinsic Coporate Website: *https://intrinsic.com*.

[7] IRC chat log with Helium engineer: *https://bot.tockos.org/tockbot/tock/2017-11-14/?msg=40209&page=1*.

[8] Levy, A., Campbell, B., Ghena, B., Giffin, D.B., Pannuto, P., Dutta, P. and Levis, P. Multiprogramming a 64kB Computer Safely and Efficiently. *26th Symposium on Operating Systems Principles (SOSP)* (October 2017).

[9] Levy, A., Hong, J., Riliskis, L., Levis, P. and Winstein, K. Beetle: Flexible Communication for Bluetooth Low Energy. *14th International Conference on Mobile Systems (MobiSys)* (June 2016).

[10] Port of Tock to Google's Titan: *https://github.com/domrizzo/tock-hotel*.

[11] Signpost GitHub Repository: *https://github.com/lab11/signpost*.

[12] Titan in depth: Security in plaintext: *https://cloudplatform.googleblog.com/2017/08/Titan-in-depth-security-in-plaintext.html*.

[13] Tock OS Homepage: *https://www.tockos.org/*.

[14] SOSP Tutorial: Tock Operating System. *SOSP 2017*.

[15] Tock Operating System Tutorial. *SenSys 2017*.

[16] Training: Tock Embedded OS. *RustConf 2017*.