MATTHEW HICKS
MIT LINCOLN LABORATORY

ImpedimentToProgress.com
mdhicks@gmail.com

Research Statement
November 30, 2016

My research focuses on the intersection between hardware and security; this includes the security of hardware [1, 2, 3, 4, 5, 6] (i.e., malicious hardware), hardware-based systems for increased software security (e.g., hardware abstractions for efficient Control-Flow Integrity enforcement [7]), and the security implications of emerging architectures [8, 9] (e.g., a user identification side-channel created by approximate memory). From a higher level, I design and implement systems that incorporate both hardware and software (sometimes even the compiler [10]) to solve challenging problems. I believe that a critical product of my research is a physical system that I can share with the rest of the community to increase adoption of my ideas and to help other researchers make progress [11].

My dissertation [12] focused on how to deal with processor imperfections, both intentional and unintentional. During my postdoctoral research, I continued to create new systems for tolerating both intentional and unintentional processor imperfections. I also expanded my research to investigate the security implications of new architectures, including approximate hardware and densely packed DRAM (rowhammer). Lastly, while a postdoctoral researcher, I started to explore ways to leverage novel hardware mechanisms to make software more secure.

My research agenda going forward consists of three broad topics, each broken down into several projects (some immediate and some years away). The first topic is a how to design and manufacture hardware with hardware-level adversaries in mind. The central thread in this line of research is the microkernelization of hardware: decompose currently monolithic hardware into layers defined by the security properties required. The layers fit naturally with trends in split manufacturing and 3D-IC and can leverage my past research on in-hardware invariant monitors [4, 5]. The second topic is exploring how hardware intended to increase system security ages. The observation driving this line of work is that, unlike software, hardware ages. Researchers tend to analyze their hardware proposals only at one point in its life cycle—the beginning. Unfortunately, hardware may age in a way that reduces security. Lastly, I am interested in tackling problems beyond hardware security [10, 13]. I believe that we are at the forefront of a wave of computational devices that replace burdensome batteries with harvested energy. I will explore ways to make existing programs run on harvested energy without burdening programmers.

In the following sections: I describe the core of my research philosophy, followed by sections outlining my past and future work in my core research thrusts, and conclude with a note about collaborations.

1 Research Philosophy

Successful research impacts the lives of those outside of academia. Achieving such impact requires addressing "big problems" that impact society and being adopted by industry. I care about addressing the problems that act as impediments to progress (hence my website name). By targeting larger, more complex and general problems, I have been able to publish at the top conferences that cover broad areas (e.g., Oakland for Security and ISCA for Architecture). Targeting top conferences increases the likelihood of adoption by forcing me to focus on "big problems", raising my research to a higher bar, and exposing it to the widest possible audience.

To increase the chances of industry considering my research, I believe that I must reduce—as much as is practical—their hurdles in accepting my ideas. The key to this is building experimental prototypes as close as possible to deployable systems and vetting those prototypes using a diverse set of realistic workloads. Since my research spans both hardware and software, I often implement my designs on an FPGA or ASIC and test using industry standard benchmarks running on top of Linux. While going to these lengths to demonstrate the effectiveness of an idea does slow the pace of my research, I believe that the increase in quality that implementing my ideas on real platforms brings is worth the sacrifice.

Another opportunity for impact comes from contributing to open source. I make it a goal for any project that I work on to, upon publication, open source the system, benchmarks, and scripts for that project. For example, the source code for my paper that presented the first detection algorithm for design-time malicious circuits UCI [1] has been used by researchers inside academia [3], researchers in industry, and even by consulting firms working with the military. Besides open sourcing my code, I have had the opportunity to contribute to existing open source projects. As a part of completing my dissertation work [12], which involved architectural modifications to the OR1200 open source processor; I contributed my fixes back to the OR1200 RTL, the bootstrap code, and to the Linux port. In a more recent project on identifying and detecting security-critical processor errata SPECS [4], I also contributed updates to the Open Verification Library that, for the first time, allowed assertions in the library to be synthesized to real hardware.

2 Ongoing Research Thrust: Hardware Security

2.1 The Security of Hardware

As researchers devise ways to better secure software, they force attackers to target hardware to gain control of victim machines. Hardware is an ideal place to attack because it is difficult to patch and attacks in hardware compromise the entire system—even otherwise secure software. A large part of my research career has focused on creating hardware attacks and developing mechanisms for detecting those and existing hardware attacks. I am also interested in the general problem of bugs in hardware, because, as shown in SPECS [4], even unintentional bugs can serve as footholds for malice.

Detecting design-time malicious hardware - Oakland 10 [1]: This project was the first attempt to detect malicious hardware at design time—i.e., security verification. This project presented UCI, a malicious hardware detection algorithm that looks for unused circuits in a design by inspecting circuit data values during simulation. This project also produced an automatic suspicious circuit removal tool BlueChip that physically disconnects potentially malicious circuits at design time and uses software-level simulation to route execution around those disconnected circuits at run time. UCI spawned several other design-time malicious hardware detection techniques (presented at top-tier security conferences).

Defeating design-time malicious hardware detection techniques - Oakland 11 [3]: Given the success of UCI at detecting known malicious hardware, the next question to answer is whether it is possible to detect all possible malicious circuits in hardware at design time. This project set out to systematically validate the UCI algorithm. This project was the first to formalize what it means for a circuit to be malicious. This made it possible to construct arbitrary malicious circuits in a systematic fashion to see if any evade UCI detection. This project showed that it is generally possible for attackers to fold malice into a circuit—as opposed to the malicious circuit being a circuit of its own—hiding malicious insertions from design-time analysis tools like UCI. These results showed, for the first time, that claiming a design malicious circuit free is akin to saying that it is bug free.

Defending against deployed malicious circuits - On going [5]: The previous project showed that finding all malicious circuits in a design is impractical (given an informed attacker), thus the threat of deployed malicious circuits is real. This project sets out to create the first system for protecting processors from deployed malicious circuits. Responding to an unknown threat requires a fabric capable of being reprogrammed, based on the discovered threat. The fabric proposed in this project enables hardware manufactures to express assertions on hardware state. The system that I propose monitors these assertions at run time—i.e., dynamic verification.

Identifying and detecting a subset of processor errata as security-critical - ASPLOS 15 [4]: Motivated by my dissertation work that showed that general processor bug patching approaches are not practical in real-world usage, this project argues for a more targeted approach. This project proposed dividing processor bugs into two classes: security-critical and general. My insight was that there is a core set of processor functionality that software must depend on and any bugs in that subset of processor functionality could result in security compromises. Another insight is that there are many existing, software-based approaches for verifying that the general functionality of a processor is correct, thus there is no need to support patching it in hardware (as opposed to previous processor patching approaches). This enabled me to create lightweight, in-hardware detectors that monitor updates to security critical state. To identify a security-critical subset of functionality, I examined the errata of real processors. To validate the detectors, I ported the mined errata to an open source processor and tested using Linux and a set of popular benchmarks. Besides being the first to propose the notion of a security-critical subset of processor functionality, this project also put forward the idea of targeted dynamic verification of processor state updates.

Analog malicious hardware - Oakland 16 [6] (Best Paper Award): UCI inspired many design-time malicious hardware detection approaches. All of the approaches have one thing in common: they work at the digital level of abstraction. Limiting looking for malicious circuits to digital circuits is fine when all of the known malicious hardware also works at the digital level. This project defeats all known design-time defenses to malicious hardware by implementing the attack trigger at the analog level. This line of research contains two types of attacks: one that relies on circuit aging and one that use added capacitance to siphon charge from a victim toggling signal; both triggers are analog representations of counter-based triggers. Being implemented using analog components, they can express in a single gate's area what would take 100's of gates for a digital circuit—and the change does not exist at the digital level, so it is not able to be simulated. Because of this, evaluating these attack required fabricating the first malicious processor.

2.2 Hardware for Security

As a postdoctoral researcher, I was able to investigate using hardware to increase system security. Hardware is the lowest layer of the system and monitors implemented in hardware can inspect system state every cycle. There are two challenges common to hardware additions targeted at increasing software security: (1) limiting hardware overhead, which can result in slower execution and increased power consumption and (2) bridging the semantic gap.

Secure return address stacks - Submitted [7]: One popular technique for increasing software's security is to enforce control-flow integrity. Enforcing control-flow integrity prevents malicious code on the system from co-opting existing code to perform malicious functionality. For this project we motivate the elimination of returns from the instruction set, instead, we enforce a policy that a callee must pass return to its caller. Hardware is the ideal layer to implement such a policy because it is protected from all software—even a malicious operating system. The cost for such protections is low: a small buffer for return addresses and a private area in memory for buffer management firmware to run. The challenge is the semantic gap: separating operating system from user mode code, knowing what process is executing, and dealing with threads. Building a realistic prototype that runs Linux also brings the challenge of non-standard control flows, e.g., setjmp-longjmp.

In-hardware malware detector - On-going: Anti-virus tools have been around for decades, but most computer savvy users do not use them. The reasons many avoid anti-virus tools are system slowdowns and false positives. An even more serious problem with existing anti-virus tools is that they are implemented in software, making them vulnerable to other software on the system, such as root kits. This project seeks to implement an in-hardware anti-virus tool that samples key hardware state (e.g., changes in privilege mode and diversity of outgoing packets) every clock cycle. Each untrusted computer on the network then sends this data to a trusted computer in the network-operations-center (NOC) over a secure chip-to-chip network. At the NOC, complex machine learning algorithms use the data to determine the health of the network. Being in hardware eliminates the vulnerability and slowdowns of software-implemented anti-virus tools, while shifting the burden of system security from many non-expert users to a handful of experts in the NOC.

2.3 Security Implications of Emerging Hardware

An exciting area of security is in emerging architectures. Every year, architects release a bevy of new systems. The systems are often analyzed for how they perform and how much they cost, but infrequently analyzed for their impact on system security. I take an orthogonal view of new architectures: I looking for ways they may be misused or how they compromise users even if used correctly.

Security implications of approximate memory - ISCA 15 [8]: Section 3 details a project that was the first work to uncover the impact on security of emerging approximate hardware. Specifically, it exposes how existing approximate memory proposals actually create a system identifying side-channel due to the unique and repeatable error pattern in DRAM.

Protecting commodity systems against rowhammer attacks - ASPLOS 16 [9]: DRAM increases in density with every generation. As cells in DRAM get closer together electrical interference increases quadratically. With DDR3, the interference became so high in some chips that special access patterns could flips bits in adjacent rows to those being accessed (rowhammering). Researchers showed how attackers could weaponize such behavior to enable malicious software to take control of a victim machine with susceptible DDR3. While researchers have proposed hardware modifications to prevent these types of bit flips, there was a need for a system that protected existing systems. To address that need, this project represents the first general-purpose software solution to the problem: re-purpose existing in-hardware event counters to detect behavior indicative of a rowhammer attack (or a really poor performing cache eviction policy). In the event of a suspected rowhammer attack, being in software enables a range of responses from refreshing adjacent rows to moving the rows accessed in the attack to an area of memory where the adjacent rows are empty or not vulnerable to rowhammering.

3 Emerging Research Thrust: Approximate Computing

The research directions presented so far focus on the security of the hardware platforms of today, what about tomorrow's platforms? I am interested in analyzing the security implications of those new architectures. I believe that it is crucial to include security considerations when developing new technologies—not after the fact. Otherwise, you put end users at risk and limit adoption of new technologies. One resurgent area of research is approximate computing. Approximate computing targets applications that do *not* require exact results. The idea behind approximate computing

is to trade accuracy of computation for energy savings or performance benefits. Approximation techniques exist in software (e.g., loop perforation), but there is growing interest in hardware support for approximate computing.

In Probable Cause [8], I go beyond the performance implications of approximate hardware and examine its security implications. Specifically, I examine a new wave of systems that use approximate DRAM to reduce system power and increase memory throughput. Probable Cause shows that approximate memory leaks identifying information about the host system. As a part of this research, I built the first approximate memory system using SDRAM chips and a microcontroller [11]. To create approximate memory, I disable memory refreshes between writing the initial data and reading what data remains after a given delay. By changing delay and temperature, and experimenting with multiple SDRAM chips, I was able to produce the first model of how approximate memory behaves. I then use this model to analyze how approximate memory effects applications on commodity hardware—showing how running on approximate memory fingerprints users.

I am interested in two directions for future research dealing with hardware approximation: (1) the security implications of other forms of approximate hardware and (2) the end-to-end effects of approximation. For security, the driving observation is that hardware, in general, has a varying response to its environment. For example, some logic gates have a lower threshold for a voltage that it reads as a `1` than other logic gates. In Probable Cause, experiments showed that for DRAM the relative decay time (i.e., variation) for a memory cell was determined largely by manufacturing time variations—thus it is unlikely to change, which makes it a signature. I am interested in exploring if this pattern holds for other types of hardware.

Lastly, I am interested in investigating the end-to-end effects of approximation. Current proposals for hardware approximation are evaluated in a sandbox; there is no work that explores the compounding effects of different sources of approximation. My intuition is that each source of approximation affects bits in a data word with different probability and that each source of approximation has a different curve. The desired outcome is that some sources of approximation work well together because they produce errors in the same locations, while other sources lead to system failure because they produce errors in different bits of a word.

4 Emerging Research Thrust: Power-dictated Computation

The miniaturization of compute power moves us closer to the realization of computational dust; the limiting factor is now the source of power—no longer speed or efficiency. Thus, it is time that power dictates what computation gets completed as opposed to the current model that focuses on reducing power for a given—fixed—computation. I refer to this line of research as power-dictated computation. In a power-dictated computation model, the availability of power dictates the duration and intensity of computation. This requires that computation respond to power availability in a fluid-like manner. Having devices capable of such fluid-like computation enables a host of deeply embedded, energy harvesting applications not possible today. The central challenge is current models of computation assume continuous power; i.e., programs fail to execute correctly when their state gets contaminated due to frequent power resets.

Batteries in Internet-of-Things class devices will be the first casualties in the move towards power-dictated computation [14]. My first projects in this area aim to create an automatic and lightweight checkpointing system that continuously saves the state of software to stable (i.e., non-volatile) storage. The compiler adds the checkpointing and restart routines, so the systems apply to existing software and there is no need for programmers to reason about the effects of frequent power cycles. The challenge is finding a checkpointing mechanism that is lightweight, because software state needs to be backed-up continuously due to the unpredictability of power failures with harvested energy. The key to this is in extending idempotency to non-volatile memory. Idempotency is a property that a sequence of instructions has as long as it never overwrites values that it previously read from. My early results show that it is possible to decompose automatically programs into a series of checkpoint-connected idempotent sections, either using only a compiler [10] or using small architectural modifications [13].

While the initial projects look at compiler-support and architectural support in isolation, I envision follow-on projects that look at combining compiler and architectural support to further optimize performance and enhance functionality. In addition to direct follow on work, there is the opportunity for collaboration as other researchers in the space have started to focus on idempotence. One compelling line of future work is the redesign of algorithms and communication protocols to make them robust against unbounded delay and disruption. Exploring the tradeoff space between low-level guarantees provided by the processor/compiler and increases in complexity needed to make programs robust against frequent resets will lead to a variety of novel systems.

Lastly, I have a much longer-range vision of applying the previous techniques to commodity systems. I envision server rooms of the future full of systems that support the power-dictated model of computation. This allows datacenters to fluidly scale computation resources in response to factors such as load, power availability, and thermal load. This vision is supported by the move to non-volatile main memory as seen in Intel and Micron's recently announced 3D XPoint memory.

5 Collaborations

Collaboration is a critical part of research: collaboration allows researchers to combine their effort and expertise to tackle more significant problems than any single researcher can. During my three years at Michigan I formed collaborations across areas (Architecture, Security, Systems, and Machine Learning) and across divisions (Computer Science and Electrical Engineering). I have also formed collaborations both within and outside of Michigan. Working in the government sector has opened the door to a new set of collaborators. I regularly interact with researchers inside Lincoln Laboratory and other government labs as well as decision makers in organizations under the Department of Defense umbrella. The majority of these collaborations exist outside of my PhD or postdoctoral advisors.

References

[1] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically," in *Symposium on Security and Privacy*, Oakland, pp. 159–172, 2010.

[2] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an Untrusted Computing Base: Detecting and Removing Malicious Hardware Automatically," *USENIX ;login*, vol. 35, pp. 31–41, December 2010.

[3] C. Sturton, M. Hicks, D. Wagner, and S. T. King, "Defeating UCI: Building stealthy and malicious hardware," in *Symposium on Security and Privacy*, Oakland, pp. 64–77, 2011.

[4] M. Hicks, C. Sturton, S. T. King, and J. M. Smith, "SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, pp. 517–529, 2015.

[5] C. Sturton, M. Hicks, S. T. King, and J. M. Smith, "MAGICCARPET: Verified Detection and Recovery for Hardware-based Exploits," tech. rep., University of Pennsylvania, March 2015.

[6] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog Malicious Hardware," in *Symposium on Security and Privacy*, Oakland, pp. 18–37, 2016.

[7] N. Dautenhahn, L. Mogosanu, and M. Hicks, "Nested-Flow Integrity: An Execution Spacetime for Warping Attack Geometry," in *Under review*, 2017.

[8] A. Rahmati, M. Hicks, D. E. Holcomb, and K. Fu, "Probable cause: The deanonymizing effects of approximate dram," in *International Symposium on Computer Architecture*, ISCA, pp. 604–615, 2015.

[9] Z. B. Aweke, S. F. Yitbarek, R. Qiao, R. Das, M. Hicks, M. Seaborn, Y. Oren, and T. Austin, "ANVIL: Software-Based Protection Against Next-Generation Rowhammer Attacks," in *International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS, 2016.

[10] J. V. D. Woude and M. Hicks, "Intermittent computation without hardware support or programmer intervention," in *Symposium on Operating Systems Design and Implementation*, OSDI, pp. 17–32, 2016.

[11] M. Hicks, "Public open-source repositry of physical artifacts produced by my research." https://github.com/impedimentToProgress.

[12] M. Hicks, *Practical Systems for Overcoming Processor Imperfections*. PhD thesis, University of Illinois Urbana-Champaign, April 2013.

[13] M. Hicks, "Clank: Architectural support for intermittent computation," in *Under review*, 2017.

[14] U. of Michigan, "Umich moo." https://spqr.eecs.umich.edu/moo/.