> "Security engineering is about building systems to remain dependable in the face of malice, error, or mischance. As a discipline, it focuses on the tools, processes, and methods needed to design, implement, and test complete systems, and to adapt existing systems as their environment evolves."
>
> Ross J. Anderson.

Introduction

I am a systems security researcher with the goal of building secure, trustworthy, and resilient systems. David Wheeler famously stated that "all problems in computer science can be solved by another level of indirection." Indeed. Unfortunately, despite making systems more reliable and easier to use, each layer of indirection creates an exponential number of security vulnerabilities—because protection goals must be expressed, translated, and enforced at every layer. Consequently, systems have proven susceptible to both accidental (reliability) and deliberate (attacks) failures that endanger critical infrastructure such as power grids, personal financing, autonomous cars, and even nuclear reactors.

My primary focus is the *representation* and *protection* of information as it traverses and is exposed to diverse hardware and software layers. For example, a cryptographic key stored in main memory is directly exposed to privileged software layers (*e.g.,* operating system, virtual machine monitor), or when placed in hardware caches, is indirectly exposed to unprivileged software through microarchitectural side-channels. My particular expertise is in understanding the complex abstractions (or lack of) amongst computing elements to identify pressure points giving way to fundamental security threats, and solving the problems right at those cross-layer points. As such my research engages in diverse technologies including operating systems design and implementation, compiler construction, program analysis, and microarchitectural modifications. To date, my research has addressed fundamental abstraction gaps in protection facilities for popular commodity operating systems (OSs), as well as over-exposure of control information that enables software hijacking. In the future, my goal is to 1) *micro-evolve* monolithic systems through automated decomposition for formal verification and incremental hardening; and 2) modify hardware/software boundaries to a) optimize emerging safe-programming paradigms and b) explore new zero-kernel OS designs.

Approach

Abstraction gaps arise when information and protection policies do not translate across boundaries. Solving these problems requires *holistic security*, or as Ross Anderson phrases it "security engineering", and must consider the entire variety of systems layers (including humans). My primary focus is finding, understanding, and solving fundamental gaps between high-level security policies and the ways in which trusted, low-level components fail to systematically enforce them. Commonly, security violations arise out of the unanticipated interactions between the many elements and ways attackers abuse them, as such my investigation typically includes real artifacts, which requires significant effort so as to expose subtle interactions and missing assumptions. I also believe in transitioning knowledge to practice and when possible, release source code, which strengthens the research community and the spread of ideas. In exploiting cross-layer solutions I have built and released experimental systems with many tools and in many environments (*e.g.,* LLVM, FreeBSD, Linux, Chrome, Firefox, Opera, Pin, Qemu). The result is expertise in domains ranging from operating systems, program analysis, and invariant specification for OS security [1, 2] to microarchitectural support bridging the gap between memory safety and hardware behavior under attack [3]. I also focus on identifying simple and fundamental security properties, going so far as to understand each computing element and using solutions that can most simply enforce them. I believe that security demands simplicity through modularity and encapsulation—selecting the right cross-layer abstractions for the problem—requiring not only deep expertise of all the parts but also time for incubation. In developing large systems across disciplinary boundaries I learned the value of and appreciate strong collaborations. Diversity in each researcher's experience, thought process, and cultural paradigms all added to the richness of my research. As such I have developed an approach of pursuing collaboration with experts who's keen insight exponentiates that of my own.

Research

A *secure* system prevents intrusions, and *hardening* is the process of making existing systems more secure. A *trustworthy* system is one that will not fail. *Resiliency* describes a system's ability to withstand attacks and maintain certain levels of trustworthiness even when compromised. In general, systems are complex and are forced to trust overprivileged and opaque components, where typically, problems arise when components expose more information than needed (poor

encapsulation across boundaries) or not enough (no transparency for verification or auditing). The worst vulnerabilities reside in lower-layers (both software and hardware) because they compromise all security of upper layers when exploited. In this way commodity systems software is like a modern day Titanic, where the external shell is brittle and susceptible to penetration and once penetrated instantly sinks the whole ship.

Abstractions for Operating System Hardening and Resiliency The primary impact of my research has been in the development of abstractions, mechanisms, and policies for securing commodity monolithic OSs. OSs are hard to secure because they operate at the highest privilege level, which means that standard protection mechanisms and common assumptions of user level approaches are no longer available. The primary insight of my thesis [4] was that safety critical runtime state (access control policies) resides in both memory and the CPU, and that by controlling the virtual-to-physical translations along with what machine instructions a component is permitted to execute, an elemental security kernel can be nested within the system's single-address space. My initial efforts explored the use of a novel compiler based approach that inserted a thin layer providing an abstract software interface to the hardware, effectively adding a managed language runtime to the OS. Along with my collaborators, I applied software fault isolation and control-flow integrity mitigations to secure [5] and constrain [6] the FreeBSD OS.

The Nested Kernel, my primary thesis contribution, sought similar goals but diverged from these efforts by eschewing full system analysis and complete hardware abstraction requirements and defined a new minimal operating system organization that provided services for OS developers to protect their own data. Beyond the kernel organization, the prototype demonstrated one of first single privilege level isolation techniques [1]. The core insight was to isolate the memory management unit (MMU) using the MMU, protecting memory resident page tables with read-only translations and CPU control state with instruction capabilities. The approach virtualizes supervisor privilege to allow both trusted and untrusted components to operate at the same hardware privilege level. *MMU virtualiztion*, as I later called it [7] (a term picked up by others [8]), can be used to provide fine-grained protections in the rest of the system, such as write-protection services [1] and kernel hardening against code-pointer corruption [9]. Along with collaborators, I also demonstrated Nested Kernel portability to both alternative hardware architectures and systems by implementing Nexen, Xen modified with the Nested Kernel [7], where we used it to provide lightweight sandboxing of per-VM state to constrain or eliminate 107 out of 144 Xen Security Advisories. Other researchers have further demonstrated OS and architecture portability by deploying to alternative OSs (Linux and Android) and using other hardware isolation environments (Arm, Intel VT-x, and in root hypervisor mode).

In an effort to understand how to automatically derive least-privilege compartments, I have developed μSCOPE [2], which is revealing strong natural separation in Linux, which could otherwise be considered a complex mesh of an OS. More recently I developed methods for automatically compartmentalizing legacy systems, which a student has taken into their domain of expertise (third party application environments (*e.g.,* JavaScript, Python, *etc.*)), and developed a novel strategy for decomposing and enforcing new module based policies [10]. Both Nexen and BreakApp sandboxing are some of the first approaches that detect and prevent challenging denial of service attacks.

This work has had external impact. Developers from HP labs have independently built a Linux instance of the Nested Kernel [11]. HardenedBSD, a derivative of FreeBSD, has shown interest and development in their source tree is ongoing [12]. Several core Apple kernel developers have queried about a few specific performance scenarios and applications of the Nested Kernel, suggesting specific future directions, which I continue to grow. For μSCOPE, I developed a large-scale Linux kernel object tracing system called Memorizer, which has been included in the LinuxKit (Docker Kernel Project) system [13]. I have also aggressively pursued industry adoption and been invited to talk with FreeBSD, Cisco, and IBM. In addition to industry impact, a research ecosystem founded on the basis of MMU virtualization is growing. It is now expected that kernel hardening techniques include MMU virtualization to protect against MMU remapping and disabling attacks. As a result, I have joined with other researchers to develop LINX, the LInux Nested kernel eXecution infrastructure. We are extending LINX to support efficient fine-grained separation to explore context-based enforcement of OS access control frameworks (*e.g.,* SELinux).

Abstractions for Hardware-Assisted Safe Execution One of the most common abstractions is the stack, however, the combination of unrestricted microarchitecture with memory unsafe C creates not only a systemically exploitable abstraction gap, but allows attackers to create Turing complete virtual instruction sets out of existing code. In developing several control-flow integrity systems I observed that the common pattern for subroutine completion was to return to the most recent procedure, *even when traversing across privilege boundaries and between threads*. I grew this insight into a multi-threaded, full system control flow paradigm, Nested Flow Path (NFP), and a novel hardware/software protection system, eXecution Graph Path Security (XGPS), that automatically enforces the NFP, deterministically eliminating one of the most pervasive threats to modern systems (return-oriented programming) [3]. Out of this project my student and

I recognized that a similar hardware mechanism could efficiently and simply capture a large, emerging class of memory safety mitigations (safe region), and grew the idea to provide a new memory abstraction, MicroStache, that combines with static checks to provide zero-cost runtime protection [14].

Abstractions for Deterministic Replay One critical requirement of resilient computing is the ability to audit and recover from policy violation. However, non-determinism in the form of system inputs and microarchitectural races create opaque boundaries hindering forensics. Deterministic record and replay addresses this problem by recording execution with sufficient fidelity. A key abstraction I developed is the *instruction footprint* invariant that specifies all inputs as CPU or memory values, which made it simpler to understand, describe, and implement non-atomic instructions that required micro-operation emulation. I used this insight to implement a replayer, PinCap, that validated the Intel QuickRec FPGA prototype [15]. PinCap proved useful immediately, as I used it to expose and solve hardware bugs by examining branch decisions and work completed in replayed executions. This work also resulted in the first relaxed consistency memory model patent for emulating Total Store Order interactions from a real Intel x86 CPU, where I used Pin to virtualize main memory and hold virtual CPU internal register state [16].

Abstractions for Automatic Browser Security Like OSs, web browsers are complex and create overprivileged environments lacking encapsulation. For example, despite being a memory safe language JavaScript allows any arbitrary code to access and modify all environment state, such as overwriting functions. I developed security retrofitting techniques to apply new security mechanisms automatically [17] and multiplex web requests in three browsers to detect and avoid exploits in any single one [18]. Collaborators and I are exploring how to automatically decompose and protect application and browser state from over-privileged components with μSCOPE.

Future Work and Vision

My completed work demonstrates that it is feasible to retrofit real protection into existing monolithic designs without requiring new hardware or sacrificing too much performance. This is only a small part of the larger *micro-evolution* of monolithic systems towards decomposed, modularly encapsulated systems. My overarching thesis is that we can make trustworthy system design feasible, including formal modeling and verification, through simplicity and automated program analysis and transformation. I believe this can apply not only in building new systems but also in retrofitting good security design into existing systems, continuously evolving the old into the new. The work has already begun with the REVOLVER grant, spearheaded by myself and supported by the Office of Naval Research (BAA #N00014-17-S-B010). A second research thrust is also underway to investigate novel hardware abstractions for safe and secure computing. The goal of this work is to explore hardware/software co-design to find the best abstractions for emerging compiler-assisted protections and to explore the potential of a zero-kernel system using hardware-based instruction capabilities.

In general though, systems security research is at an inflection point. Early explorers covered a tremendous amount of ground, peering far into the future, but the technology was lacking, severely limiting its practical consideration. As hardware and software toolchains continue to advance, many of the old questions can be re-explored with new opportunities: specifically at abstraction points. We also have one thing that our founders lacked: experience in how systems are used and have evolved over decades, providing insight and guidelines for what support is required. Instead of proposing approaches to guess at the best abstractions (for programmability, efficiency, etc.), we have the opportunity to optimize for dominant uses.

Micro-Evolution: Separation for Least-Privilege, Auditing, and Verification in Evolving Systems Complexity is one of the most challenging issues in computing and is a primary reason we have large overprivileged environments. In the area of security there have been several powerful approaches, like least privilege and verification, but no *systematic* application. The reality is that, for most applications, security will always be an afterthought (except for a few specialized applications) because if no one uses the system then there will be no information to protect. Moreover, predicting the way access controls will evolve is impractical to handle without aid.

The only way to build trustworthy systems under this complexity is to use the age old principle of *divide-and-conquer*: decompose systems into simpler encapsulated variants with explicit interactions. Once this is accomplished we can then apply analysis techniques to understand and control the interactions. Furthermore, we can apply formal modeling and verification to assert functional correctness of smaller modules, as well as specify and prove global policies. With proper runtime enforcement and reference monitors we can also turn default *allow-all* environments into *deny-all* and audit any violations.

However, such a process will require systematic techniques to decompose applications in ways that isolate privilege while not disrupting normal operation. This becomes more challenging when considering that most code is not written by any single person, and therefore no one is an expert. And, the amount of code is so large that decomposition may demand far greater effort than building from scratch. The code is also likely to change at a rapid pace.

To address these problems, I propose a general purpose privilege separation methodology, similar to an optimizing compiler, that will mechanize policy inference using both static and dynamic program analysis. This will combine with automated transformation mechanisms and runtime monitors that enforce policies while also providing opportunities to learn and evolve the system. Once a system is separated, security sensitive components can be hardened using language specific techniques (such as memory safety or control-flow integrity), as well as incremental replacement of C code with safer counterparts. Last, I intend to pursue formal verification of the system from the ground up (with expert collaborators), starting with MMU virtualization properties and separation logic for its memory isolation and building layer by layer on top of that. Included in this verification will be a modeling element that will allow explicit reasoning about the interactions amongst components for identifying best global policies (*e.g.,* do not leak the keys on the network) and assessing the system's trustworthiness given exploitation in isolated parts of the system. Key elements of such an approach include a privilege representation that is low-level enough to map to systems layers for enforcement but expressive enough to capture high-level notions of privilege. Additionally, techniques to translate policies between layers of the stack will be necessary to ensure vertical policy integration. Work has begun along these lines with the development of an abstraction for representing generic privileges (*e.g.,* modify or access) and analysis of large software systems to explore both the potential of least-privilege and validate the representation's generality. Work has also begun on automated transformation and enforcement mechanisms in both Linux and JavaScript environments.

Although the work is in its infancy I believe it is going to provide the substrate with which to fundamentally shift the development and structure of software through new principled security engineering techniques. The infrastructure developed also has a great potential for fine-grained provenance, which seeks to build dependency information for debugging and forensics. Collaboration with provenance experts has commenced. Additionally, the decomposition techniques break down privileges into local and external messages, which is the same model as networking. There are several problems that could leverage the policy learning infrastructure and algorithms, such as legacy application firewalls in large enterprise networks.

Hardware and Analysis Support for Safe Execution Paradigms and Zero-Kernel OS It is clear that hardware vendors are willing to spend silicon for security (SGX, MPX, CET, etc.). As new programming paradigms emerge, I intend to continue on my existing work. First, XGPS is a powerful, efficient, and complete context-sensitive path tracing and protection mechanism, which I exploited to enforce NFP. It also provides a compelling potential for encoding other types of context-sensitive policies, such as control-flow integrity for indirect calls and jumps, or encoding rules based on context for stack introspection or context-sensitive security policies. Second, MicroStache [14] showed a middle ground for memory safety, which has the potential to greatly simplify and improve the performance of full software memory safety techniques. Third, the dichotomy of a single superprivileged component and many unprivileged components is untenable as we move into the future of clouds and increasing amounts of data in large monolithic software ecosystems, which is perpetuated by ringed hardware architectures. An intriguing direction is to implement general and flexible instruction capabilities similar to the MMU capabilities explored in the Nested Kernel, but to eliminate the ringed abstraction of modern hardware, building an efficient zero-ring zero-kernel system with minimal rigid hierarchical privileged layers. Such a system would more suitably support microkernel designs with relatively flat privilege structure, as observed from Nexen and LINX. I anticipate this work building on existing memory capability architectures while exploiting it by both porting standard ecosystems, making them the client (Linux,FreeBSD), while also exploring new operating system designs.

Developing an Ecosystem for Discovery and Learning My primary objective as a leader of academic research is to create a rich ecosystem for knowledge discovery and student learning. This begins with cultivating creativity in identifying problems and solutions and relentless pursuit of discovery. Abstractions are fundamental in secure systems operations, and taking advantage of the decades of research to ask questions about the hardware-software boundary and which components have authority is what I see as powerful opportunities for impact. The problems and systems that I have scoped out with my original project ideas have been or are currently being worked on by faculty from five universities, as well as 6 undergraduate, 2 masters, and 7 doctoral students. One of these students just completed his dissertation, which was heavily influenced by ideas, problems, and infrastructure developed by my primary projects. I intend to continue on in this way so that I both provide opportunity for students to become independent scholars while pushing the knowledge frontier.

References

[1] **Nathan Dautenhahn**, Theodoros Kasampalis, Will Dietz, John Criswell, and Vikram Adve. Nested Kernel: An Operating System Architecture for Intra-Kernel Privilege Separation. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, (ASPLOS '15), pages 191–206, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-2835-7. doi: 10.1145/2694344.2694386. (cited by 23).

[2] **Nathan Dautenhahn**, Jai Pandey, Imani Palmer, Derrick McKee, Chris Akatsuka, Vasileios P. Kemerlis, Mathias Payer, Adam Bates, Vikram Adve, André DeHon, and Jonathan M. Smith. Under the μSCOPE: Analyzing Least-Privilege Separation in Monolithic Operating Systems. In *preparation for USENIX Security (USENIX Sec)*, 2018.

[3] **Nathan Dautenhahn**, Lucian Mogosanu, Matthew Hicks, and Lucas Davi. The Missing Context: An Execution Spacetime for Warping Attack Geometry. In *submission to European Conference on Computer Systems (EuroSys)*, 2018.

[4] **Nathan Dautenhahn**. *Protection in Commodity Monolithic Operating Systems*. PhD thesis, University of Illinois at Urbana-Champaign, August 2016. Advisor: Vikram S. Adve.

[5] John Criswell, **Nathan Dautenhahn**, and Vikram Adve. KCoFI: Complete Control-Flow Integrity for Commodity Operating System Kernels. In *Proceedings of the 2014 IEEE Symposium on Security and Privacy*, (SP '14), pages 292–307, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-4686-0. doi: 10.1109/SP.2014.26. (**cited by 108**).

[6] John Criswell, **Nathan Dautenhahn**, and Vikram Adve. Virtual Ghost: Protecting Applications from Hostile Operating Systems. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, (ASPLOS '14), pages 81–96, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2305-5. doi: 10.1145/2541940.2541986. (**cited by 82**).

[7] Lei Shi, Yuming Wu, Yubin Xia, **Nathan Dautenhahn**, Haibo Chen, Binyu Zang, Haibing Guan, and Jinming Li. Deconstructing Xen. In *24th Annual Network and Distributed System Security Symposium*, (NDSS '17), San Diego, CA, USA, 2017. The Internet Society.

[8] Adrian Colyer. Review: Deconstructing Xen, 2017. URL `https://blog.acolyer.org/2017/03/16/deconstructing-xen/`.

[9] Lucian Mogosanu, Adrian Dobrica, Volodymyr Kuznetsov, George Candea, and **Nathan Dautenhahn**. KASM: Precise Protection of Operating System Kernels Against Control-Flow Hijacks. In *preparation for USENIX Security (USENIX Sec)*, 2018.

[10] Nikos Vasilakis, Ben Karel, Nick Roessler, **Nathan Dautenhahn**, André DeHon, and Jonathan M. Smith. BreakApp: Automated, Flexible Application Compartmentalization. In *To Appear in 25th Annual Network and Distributed System Security Symposium*, (NDSS '18), San Diego, CA, USA, 2018. The Internet Society.

[11] Theo Koulouris theo.koulouris@hpe.com Chris Dalton cid@hpi.com, Nigel Edwards nigel.edwards@hpe.com. Split kernel, 2017. URL `https://github.com/linuxkit/linuxkit/tree/master/projects/okernel`.

[12] Oliver Pinter and Shawn Webb. Hbsd, 2016. URL `https://github.com/HardenedBSD/hardenedBSD/tree/hardened/9/kernsep`.

[13] **Nathan Dautenhahn**. Memorizer, 2017. URL `https://github.com/linuxkit/linuxkit/tree/master/projects/memorizer`.

[14] Lucian Mogosanu, Ashay Rane, and **Nathan Dautenhahn**. MicroStache: A Hardware Enforced Abstraction for Sensitive Data Isolation. In *submission to European Conference on Computer Systems (EuroSys)*, 2018.

[15] Gilles Pokam, Klaus Danne, Cristiano Pereira, Rolf Kassa, Tim Kranich, Shiliang Hu, Justin Gottschlich, Nima Honarmand, **Nathan Dautenhahn**, Samuel T. King, and Josep Torrellas. QuickRec: Prototyping an Intel Architecture Extension for Record and Replay of Multithreaded Programs. In *Proceedings of the 40th Annual International Symposium on Computer Architecture*, (ISCA '13), pages 643–654, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-2079-5. doi: 10.1145/2485922.2485977. (cited by 23).

[16] **Nathan D. Dautenhahn**, Justin E. Gottschlich, Gilles Pokam, Cristiano L. Pereira, Shiliang Hu, Klaus Danne, and Rolf Kassa. Mechanism for facilitating dynamic and efficient management of instruction atomicity violations in software programs at computing systems, Noermber 22, 2016. U.S. Patent Number 9,501,340. Filed Mar 15, 2013. Issued: Nov 22, 2016.

[17] Shuo Tang, **Nathan Dautenhahn**, and Samuel T. King. Fortifying Web-based Applications Automatically. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, (CCS '11), pages 615–626, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0948-6. doi: 10.1145/2046707.2046777. (cited by 28).

[18] Hui Xue, **Nathan Dautenhahn**, and Samuel T. King. Using replicated execution for a more secure and reliable web browser. In *19th Annual Network and Distributed System Security Symposium*, (NDSS '12), San Diego, CA, USA, 2012. The Internet Society. (cited by 12).