An Interview with Peter G. Neumann

RIK FARROW

Peter G. Neumann is Senior Principal Scientist in the SRI International Computer Science Department, where he has been for 45 years. His research has been concerned with computer systems and networks, trustworthiness/dependability, high assurance, security, reliability, survivability, safety, and many risk-related issues such as election-system integrity, cryptographic applications and policies, health care, social implications, and human needs—especially those including privacy. Currently, he is Principal Investigator of the joint SRI/Cambridge project relating to the CHERI system. He was at Bell Labs in Murray Hill, New Jersey, where he was heavily involved in Multics development. He has AM, SM, and PhD degrees from Harvard, and a Doctor rerum naturalium from Darmstadt. He moderates the ACM Risks Digest forum (http://www.risks.org) and has been responsible for 242 "Inside Risks" columns in the *Communications of the ACM*. He chairs the ACM Committee on Computers and Public Policy. His 1995 book, *Computer-Related Risks,* is still timely. He received the National Computer System Security Award in 2002, the ACM SIGSAC Outstanding Contributions Award in 2005, the Computing Research Association Distinguished Service Award in 2013, and in 2012 was elected to the National Cybersecurity Hall of Fame as one of the first set of inductees. See his Web site, http://www.csl.sri.com/neumann, for further background and URLs for papers, reports, and testimonies. Neumann@CSL.sri.com

Rik Farrow is the editor of *;login:*. rik@usenix.org

I first encountered Peter G. Neumann at the PC party for Security in Washington, DC, back around 2000. Peter was playing a grand piano and leading a group in singing songs from Gilbert and Sullivan, Tom Lehrer, and more. I later learned that Peter can play many more instruments.

Peter and I met for lunch in 2007 in Palo Alto, not far from where he works at what used to be Stanford Research Institute and is now SRI International. I was going to speak at Apple and Google over the following week about the failure of current measures that were supposed to be making our systems more secure. Peter encouraged me, then regaled me with stories about the Multics design.

Peter has been involved in security since 1965, starting with his work on the Multics file system and overall Multics development, continuing with a provably secure operating system (PSOS). His current project involves the CHERI (Capability Hardware Enhanced RISC Instructions) hardware-software system co-design [1]).

Rik Farrow: Part of what got me thinking about you was your story about part of the design of the Multics file system: getting a small group of people in a room with whiteboards and coming up with a design.

Peter G. Neumann: The first real get-together of the Multics team (MIT, Bell Labs, and GE-then-Honeywell) took place at an AT&T training center in Hopewell, NJ, the week of Memorial Day 1965. Fernando Corbató (Corby, who led the CTSS effort), Bob Daley (who created the CTSS file system), Stan Dunten (who had done the CTSS I/O), Jerry Saltzer (just about to complete his PhD thesis, "Traffic Control in a Multiplexed Computer System," 1966), and—inspirationally—Ted Glaser from MIT (co-designer with John Couleur of the really innovative hardware; former NSA, later head of the CS Department at Case Western) and his dog, and Vic Vyssotsky, Joe Ossanna, and me from Bell Labs (BTL). We discussed the emerging independently protectable segmentation hardware architecture, the desiderata for the operating system (segment descriptors, paging, and the file system), and planning for the five Fall Joint Computer Conference papers for Las Vegas in 1965. Bob Fano and Bell Labs VP Ed David (later Nixon's Science Advisor) were assigned the introductory paper, and Bob Daley and I the file system design—which largely emerged over the summer. The papers are all on multicians.org, maintained by Tom Van Vleck.

Ted was blind since age 12 but the most far-sighted person I have ever known. His impact on Multics was holistic and enormous. The first day of our week was in fact Memorial Day, and we had to find a local restaurant that was open for lunch. The only one we could find would not allow Ted's wonderful German Shepherd into the restaurant, but we finally talked them into setting up tables outside.

RF: What happened with Multics? I know Multics has continued to be used from visiting the multicians.org site, but my recollection is that the project fell apart because of disagreements between the various parties.

An Interview with Peter G. Neumann

PGN: Bell Labs dropped out of the Multics development in 1969, when AT&T upper management realized that its declared intent that Multics would replace all computers at Murray Hill, Holmdel, Whippany, and Indian Hill could not be fulfilled on time.

Ken Thompson had joined BTL in 1967, and immediately observed that the symbolic name scheme (with dynamic linking to descriptor entries) for the file system that Bob Daley and I had designed would be great for input-output, which triggered a very nice redesign of the original Multics I/O system. As a result of Bell Labs bailing on Multics, Ken found a PDP-7 that no one was using. I remember one day when Ken came in at noon for lunch with Joe Ossanna and me, and said that he had just written a thousand-line one-user OS kernel, and I suggested he should use all of his Multics experience on multiuser multiprogramming to extend his kernel. The next day he came in with another 1000 lines. That then led to Unics (the castrated one-user Multics, so-called due to Brian Kernighan) later becoming UNIX (probably as a result of AT&T lawyers).

Multics development and maintenance continued for many years after that at MIT and at the Honeywell CISL office nearby in Cambridge. Charley Clingen headed the Honeywell Multics group, and Tom Van Vleck was heavily involved in Multics from 1966 at MIT and later moved over to Honeywell. The last Multics installation, a five-processor multiprocessor configuration, ran until 2000.

In the early 1970s there was even an effort that retrofitted multilevel security into Multics, which required a little jiggling of ring 0 and ring 1. I was a distant advisor to that (from SRI), although the heavy lifting was done by Jerry Saltzer, Mike Schroeder, and Rich Feiertag, with help from Roger Schell and Paul Karger.

The Multics hardware-software effort was seminal in pioneering Jack Dennis's notion of segmentation, with hardware-supported paging, dynamic linking, a hierarchical file system, ring structures (control hierarchies), solving the buffer overflow problem, execute-only code, pure procedure sharing, innovative file backup, and lots more. The buffer overflow problem was solved by making everything outside of the active stack frame not executable, and enforcing that in hardware.

RF: Can you tell us about your work on Provably Secure Operating System for the NSA?

PGN: Multics had a considerable influence on SRI's Provably Secure Operating System (PSOS [2]), for which the security-relevant hardware and software functionality was formally specified in a common language that we created (SpecIAL), primarily by Larry Robinson and Karl Levitt. The PSOS architecture is an early example of a hierarchically designed hardware-software system, in which each successive layer could depend only on lower layers (somewhat akin to Dijkstra's THE system [3]), but where the hardware enabled an operation at an OS or application layer to be executed efficiently as a single instruction after the descriptor table and page tables were in place. I worked on PSOS from 1973 until 1980 under a contract from the NSA. Three more years of that project supported the Goguen-Meseguer work on noninterference and early work on SRI's PVS formal verification system.

In turn, Multics and PSOS had significant influence on the CHERI that we are currently developing. In addition, the CHERI hardware supports some of the security concepts from the more recent Capsicum operating system [4])—notably, its hybrid architecture and the ability to enforce least privilege and compartmentalization.

RF: I've been reading the PSOS retrospective paper [2] and am a bit confused about what a capability is. PSOS capabilities appear associated with unique user IDs with a set of access rights. These can be copied, with restrictions, and appear to be created with hardware monotonicity that would ensure that rights could never increase.

I think I am confused because I associate capabilities with both an application and a user ID, so that a user ID doesn't have the same set of capabilities for all applications she may run. Perhaps you could explain?

PGN: You are indeed confused, perhaps because each capability system—in the past, present, and the future—tends to be slightly if not fundamentally different. PSOS capabilities were different from others because the ID of the capability was *unique* for the lifetime of that processor, and could be stored in a Multics-like directory for access via a symbolic name. There was no user ID associated, because capabilities could be shared—subject to the propagation limits. This was appropriate for the researchy hardware-software spec, as it was conceptually simple but not very practical.

CHERI capabilities [1] are more local in nature rather than global potentially for every user and every process. They are **fat** pointers, which include bounds and permissions, along with a nonforgeable tag to ensure nonforgeability of the capabilities. One common thread between PSOS and CHERI is that both have object-oriented capabilities with either default types (for virtual memory) or user-defined types (for objects).

The huge difference is that CHERI solves the legacy compatibility problem and allows crapware to coexist safely with very trustworthy operating systems, applications, compilers, and so on.

RF: The fat pointers that I know about are part of the D language extensions to C [5] and include a range with every pointer to prevent buffer overflow attacks. Can you tell me how pointers in CHERI are different?

PGN: Conventional fat pointers are typically virtual addresses that have been extended with additional metadata such as bounds and permissions. CHERI's fat-pointer capabilities add notions of sealing and unsealing (for strongly typed object capabilities), provenance (ensuring that new capabilities are properly derived from other legitimate capabilities), and monotonicity.

RF: In the PSOS paper, you describe the system as hierarchically layered, but also write that such multi-layer designs aren't found in contemporary systems. Could you explain the importance of layering and perhaps why it's not found in systems today?

PGN: Layered assurance is premised on formal analyses that can be built up layer by layer. Dijkstra's THE system [3] had informal proofs that there could be no deadlocks between layers, because the locking strategy at each layer involved purely hierarchical dependencies. Years later I asked Nico Habermann [6] about that. He said they had actually discovered a hitherto undetected deadlock within a single layer, but never any that involved multiple layers.

The Multics ring structure enabled up to eight rings, although rings 0, 1, 2 were the primary ones that were used by the system itself with ring 4 used by user software. Outer rings were left for applications. Nothing that happened in ring 1 could ever clobber ring 0, ring 2 could never clobber lower rings, and so on. Many systems have a layered structure, but typically it is only kernel and user—that is, only two layers. CHERI could implement many layers easily using the capability mechanism, either implicitly or explicitly.

PSOS had 17 layers in the conceptual architecture. Layer zero had two instructions out of which everything else was built in initialization—creating a new capability with desired privileges and creating a copy of an existing capability with at most the same privileges. That's CHERI's monotonicity property (which includes privileges and bounds that may never increase). The lowest 7 PSOS layers were intended to be implemented in hardware.

The Multics ring property is conceptually similar to the Biba multilevel integrity dependence property—that each layer (or in Biba's case, integrity level) must depend only on itself and on lower layers, at least in principle. There are of course some trusted exceptions involving calling into a lower ring—and then returning without acquiring any lower-layer privileges. CHERI explicitly introduces the principle of intentionality to counter the fact that calling something else must not allow the something else to confer properties elsewhere or usurp privileges it does not have. This addresses so-called confused-deputy attacks.

As you can see, this all fits together—from Multics to PSOS and Capsicum to CHERI, with deep awareness from my Cambridge colleagues on all of the other attempts at past and contemporaneous capability-based systems, and proactively trying to avoid the pitfalls of the past while adopting other ideas that might work in this context (such as capabilities that act as fat pointers). Robert Watson in particular has an absolutely uncanny understanding of all of this, and someone without whom we could have never gotten this far so quickly in developing CHERI.

RF: This is making sense to me. I didn't realize that the lower seven layers of PSOS were supposed to be done in hardware.

I'm glad you brought up rings, as people widely misunderstand them today. I did want to mention that virtualization actually has meant the creation of more rings, such as "ring -1" for hypervisors.

What I was wondering is whether CHERI provides a model for capabilities that would be useful for people to learn about today? I haven't finished reading the technical report yet, but it seems like capabilities are a bit like container technology, in that capabilities are used to control access to various namespaces, very much like Linux containers.

How closely does CHERI mimic the systems that Multics ran on? I realize that both the GE 645 and CHERI make use of segment registers as hardware support for isolation. That seems different from the capabilities discussed in Capsicum.

PGN: PSOS used ideas from Multics. CHERI used ideas both from Multics and from PSOS and Capsicum. But Capsicum is software only and relies on potentially untrustworthy hardware. We rectified that in CHERI, which adopted the hybrid model of Capsicum, but designed hardware that would greatly enhance the trustworthiness of operating systems and applications. It also advances operating systems beyond Capsicum. Try to understand the CHERI papers [7] as new stuff, although each paper states how we differ from the past. The tech report will help a lot. The report is long but well structured. It includes a chronological history of how we got to where we are, as well as how it relates to other efforts. All of this should be extraordinarily valuable for learning about the security pitfalls that can be overcome through enlightened hardware and total-system architecture.

RF: Does CHERI provide the same or better support in hardware than did systems running Multics? Does CHERI's hardware support extend beyond segment registers, for hardware that provides real isolation for different capabilities? Those are questions I'd like you to answer.

PGN: The Multics hardware was designed by John Couleur and Ted Glaser. John was a pure hardware person. Ted was someone who got John to build independently protectable segmentation into the hardware, with a deep understanding of how the operating system and compilers might exploit it for paging and shared pure procedure, as well as for security, reliability, robustness, resilience, and more.

The CHERI hardware ISA began with an open-source MIPS 64-bit ISA formal spec (developed by Cambridge), and added capability instructions and capability registers. It represents a complete clean-slate hardware-software co-design. CHERI has proceeded iteratively, with a few very minor but useful additions or refinements of particular instructions over the past seven years because of better understanding of the operating-system and compiler needs.

CHERI can do anything Multics could do—segmentation, paging, dynamic linking, ring-structured software—and much more (high-assurance fine-grained access controls, fine- and coarse-grained compartmentalization, e.g., within a given application or within an OS, and among all of the different applications, virtual partitions and more). We believe that we will soon have some viable approaches to the active device input-output direct memory access problems. The Multics General Input/Output Controller (GIOC) had that problem in spades, because the GIOC needed absolute memory addresses, bypassing all segmentation, paging, and memory protection. CHERI hopes to extend the reach of the capability-based protection to I/O and embedded active devices and microcontrollers.

A big difference between Multics and CHERI development is that the Honeywell 645 was pretty much frozen early in the hardware design. Getting the operating-system dynamic linking to work with the hardware took several iterations, and might have been abetted by hardware improvements that were not available. On the other hand, the CHERI ISA has been fluid and able to respond to the needs of software and compilers, as we increasingly learned how to take advantage of the CHERI capability architecture—which is somewhat different from most of the predecessor capability systems. Various instructions were added along the way to simplify software development. CHERI also adopted the PSOS idea of capabilities for typed objects in hardware (noted above), which was not possible in Multics.

There is considerable detail that we have glossed over, and other efforts such as microkernel operating systems, application trustworthiness, and the use of formal methods to ensure that the hardware ISA satisfies the required trustworthiness properties and principles. In addition to [1], see [8] and [9].

References

[1] R. N. M. Watson, R. Norton, J. Woodruff, A. Joannou, S. W. Moore, P. G. Neumann, J. Anderson, D. Chisnall, N. Dave, B. Davis, K. Gudka, B. Laurie, A. T. Markettos, E. Maste, S. J. Murdoch, M. Roe, C. Rothwell, S. Son, and M. Vadera, "Fast Protection-Domain Crossing in the CHERI Capability-System Architecture," *IEEE Micro Journal,* vol. 36, no. 6 (September-October 2016), pp. 38–49: https://goo.gl/Vu8W1J.

The current comprehensive hardware ISA document is online: R. N. M. Watson, P. G. Neumann, J. Woodruff, M. Roe, J. Anderson, J. Baldwin, D. Chisnall, B. Davis, B. Laurie, S. W. Moore, S. J. Murdoch, R. Norton, S. Son, H. Xia, "Capability Hardware Enhanced RISC Instructions: CHERI Instruction-Set Architecture (Version 6)," Technical Report no. 907, University of Cambridge Computer Laboratory, July 2017: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-907.pdf.

[2] P. G. Neumann and R. Feiertag, "PSOS, Revisited," ACSAC, 2003: http://www.csl.sri.com/users/neumann/psos03.pdf; http://www.csl.sri.com/neumann/psos/psos80.pdf (full 1980 report—scanned).

[3] E.W. Dijkstra, "The Structure of the THE Multiprogramming System," *Communications of the ACM*, vol. 11, no. 5 (May 1968), pp. 341–346; also, Wikipedia, "Edsger W. Dijkstra," section 2.5 (Operating system research): https://en.wikipedia.org/wiki/Edsger_W._Dijkstra#Operating_system_research.

[4] R. N. M. Watson, J. Anderson, B. Laurie, and K. Kennaway, "Capsicum: Practical Capabilities for Unix," in *Proceedings of the 19th USENIX Security Symposium*, August 2010; see also Capsicum Technologies: https://wiki.freebsd.org/Capsicum.

[5] Cello, "A Fat Pointer Library": http://libcello.org/learn/a-fat-pointer-library.

[6] Wikipedia, "Nico Habermann": https://en.wikipedia.org/wiki/Nico_Habermann.

[7] CHERI project home page: http://www.cl.cam.ac.uk/research/security/ctsrd/cheri/.

[8] R. N. M. Watson, P. G. Neumann, and S. W. Moore, "Balancing Disruption and Deployability in the CHERI Instruction-Set Architecture (ISA)," in *New Solutions for Cybersecurity*, ed. H. Shrobe, D. Shrier, A. Pentland (MIT Press/Connection Science, 2018).

[9] P. G. Neumann, "Fundamental Trustworthiness Principles," in *New Solutions for Cybersecurity*, ed. H. Shrobe, D. Shrier, A. Pentland (MIT Press/Connection Science, 2018).