An Interview with

PETER J. DENNING

OH 423

Conducted by Jeffrey R. Yost

on

10 April 2013

Computer Security History Project

Naval Postgraduate School, Monterey, CA

Peter J. Denning Interview

10 April 2013

Oral History 423

Abstract

This interview focuses on Peter Denning's pioneering early contributions to computer security. This includes discussion of his perspective on CTSS and Multics as a graduate student at MIT, pioneering (with his student Scott Graham) the critical computer security concept of a reference monitor for each information object as a young faculty member at Princeton University, and his continuing contributions to the computer security field in his first years as a faculty member at Purdue University. Because of an extensive, career spanning oral history done with Denning as part of the ACM Oral History series (which includes his contributions as President of ACM, research on operating systems, and principles of computer science), this interview is primarily limited to Denning's early career when computer security was one of his fundamental research areas.

This material is based upon work supported by the National Science Foundation under Grant No. 1116862, "Building an Infrastructure for Computer Security History."

Yost: My name is Jeffrey Yost and I'm here today on the campus of Naval Postgraduate School with Professor Peter Denning. This is an interview for CBI's NSF-funded project, "Building an Infrastructure for Computer Security History." It's April 10, 2013. Because of your lengthy, publicly available ACM oral history, I'll focus pretty much on the topic of computer security but I'd like to get some basic biographical information first. Can you tell me when and where you were born?

Denning: I was born in New York City, in Queens; in 1942, January 6.

Yost: And did you grow up there as well?

Denning: I was the firstborn. My family lived in Queens at the time. In 1945, we had become a four-children family and my parents moved us to a bigger house in Darien, Connecticut. I actually grew up in Connecticut. My father became a commuter, riding on the train every morning down to New York City while the children went to school in the suburbs.

Yost: And what did your father do for a living?

Denning: He was a lawyer in the entertainment industry. He worked first with RCA and then later with Universal Studios. He became a vice president for the legal department later on in his career with Universal.

Yost: You attended Fairfield Prep?

Denning: That is correct, yes. That's a Jesuit high school.

Yost: Were there subjects at that time that you were particularly interested in or had a special aptitude for?

Denning: I had always been interested in science. Since I was a small kid, I was very interested in astronomy. I read voraciously, went to planetariums, built telescopes, set out looking at the stars, memorized the constellations, did my school science projects on astronomy. I developed a side interest in plants, botany -- trees, and flowers – but not to the same depth. I could identify many trees and flowers. In the Boy Scouts I learned which ones were safe to eat and which ones weren't, in case I were ever stranded in the wilderness and had to survive on edible plants. Then somewhere around age 12 I started to get interested in electricity and electronics. That really turned me on. I gobbled up all the books I could find. I found some books that had been issued during World War II as basic electricity training manuals for soldiers, and that's how I learned my basic electricity. I became interested in electronics, radio, television and all sorts of electronic systems.

Yost: Did you do any ham radio?

Denning: I never really got into ham. I thought about it. I learned the Morse code, but just didn't want to go to all the trouble of getting the licenses. I don't know why. I had some friends who were ham radio operators but I didn't go into it myself. From Heathkits I built radios, sound systems, test instruments, photovoltaic door sensors, blinking Christmas lights, and other electronic circuits. I found myself mostly interested in the computer end of it. Computers were brand new then. They really intrigued me. The idea that electronic circuits that could do brainy things intrigued me. When I got into high school, Fairfield Prep, I was gifted to be taken under the wing of the math teacher, Ralph Money. He was the head of the science club and he encouraged many of his students to join; so in my second year I joined the science club. He was a great coach and a great mentor. He had all the guys in the club doing projects. Everybody had to do a project for the spring science fair. So we spent most of the school year preparing for the science fair. I did three science fairs while at Prep, in my sophomore, junior, and senior years. For the first, I built a computer made of relays that added numbers. I spent a considerable amount of time acquiring relays from cheap outlets for that machine. I remember finding a supplier who sold me a bunch of discarded surplus submarine relay boxes for a dollar each. Each one contained five relays. I hacksawed my way into those massively strong containers to liberate those relays. Eventually I freed all those relays from their hermetically sealed prisons. Then I discovered they used large amounts of current and I had to design an unwieldy power supply for my computer. But I was rewarded for these efforts with a first prize in the science fair for my machine! Ralph Money was very proud of me and all the other guys who did well in fair. Before we left the fair, he already challenged me to my next science project: "What are you going to do for next year?" I said, "Well, next year, I'll build a machine that solves equations." I had no idea how to do that. I plunged into a massive design project that soaked up evenings and weekends for much of my junior year. I talked a neighbor out of an old pinball machine and used its parts to built a computer to solve linear equations – those of the form A*x*+B=0. The operator would set values for A and B on the control panel, and the machine would find *x* such that A*x*+B equals zero. It did it by a simple method of successive subtraction using pinball stepping relays and it displayed its final answer on a graphics output screen from the pinball scoring relay.

Yost: Very impressive for a 17-year-old high school student.

Denning: Yes, that computer needed lots of relays, many more than in the prior computer. I couldn't use the relays from the prior computer because they used too much power and I couldn't afford the power supply. Now, a friend of my father had an old pinball machine in his basement that he had bought surplus but never used. It took me several months to convince him to free the pinball machine from its captivity and turn it into a science-fair computer. He finally agreed. And I did it. My linear-equations solver used up all the pinball parts, even the one which keeps the score. Pinball machines had a stepping relay that rotated a wheel with stenciled numbers; a light shining through a stencil told you what your score was. I put in a brighter bulb and caught the stencil image on a tracing paper – my first graphical display – that projected the value of the solution *x* on to the little screen. People loved it. They also loved the idea of that the machine was made of pinball parts. They loved how it sounded -- every bit like a pinball machine while it cranked away solving the equation. And like every pinball machine, it sparked liberally as relay contacts opened and closed, enveloping the computer in a cloud of ozone. All the clatter and odor attracted quite a crowd, which turned out to be important in getting the judges to pay attention. They came over to see what the crowd was about and we had quite a conversation about the machine. To my astonishment, I learned the next morning that my machine won the grand award of the fair. Part of my award bounty was the right to display at the New England Regional Fair at Brown University. There I met other science fair winners from all over New England. But hardly had the ozone settled, when Ralph Money said, “I suppose that your next science project will be a machine that solves quadratic equations!” And I responded, “I’ll see your quadratic machine and go for a cubic machine. My next machine will solve cubic equations!” That machine was too complicated for relays. Instead I designed circuits that represented values as voltages and mixed them in a bank of vacuum tubes to perfectly cancel when the *x* voltage solved the equation. It wasn’t perfect but it did a pretty good job solving equations. It was also perfectly electronic – no moving parts, no clicking, no sparking, no ozone. To operate the machine, you turned knobs to indicate the values of coefficients, then you turned a solution-knob until a voltmeter reached a minimum value. The position of the solution-knob was the value of *x* solving the equation. I took that machine to the third science fair, fully expecting to receive another first place for the accomplishment. Imagine my disappointment when I learned the next day that my machine was awarded only second place! I realized that I just had a lesson in marketing. The judges just didn’t understand what it did. I had done absolutely nothing to make posters about the machine, gather a crowd, or get others to market it. I just thought that a machine to solve cubic equations was so cool that everybody would be awed. But nobody noticed it. In addition, the year before Ralph Money roamed the aisles of the fair loudly proclaiming the wonders his Prep boys had brought to the conference, thus gathering crowds for the judges to notice. But Ralph Money had moved to another school that year and I had no unseen marketing agent.

Yost: You didn't have that promoter.

Denning: I've never forgotten that lesson. Today, when parents ask me how their kid can win a science fair I tell them you've got to have a cool idea, it has to deal with some issue that people care about, and you've got to market it. Even after you put it all together, you've got to figure out how you're going to get the crowds over there and how you're going to hold their attention. It is not true that if you build a cool machine people will come. Anyway, I graduated from high school with a lot of electronics expertise and some experience in designing computers. Experience enough to tell me that designing computers that work reliably is hard. I was really into computers. I had built what I could and I wanted to major in them in college.

Yost: You went to Manhattan College?

Denning: That is correct. My father wanted me to try for M.I.T. But my high-school Jesuits had convinced me that I should go to a Catholic college, not a heathen place like M.I.T. I picked Manhattan College because it had the best reputation in engineering among Catholic schools. My father was disappointed with my choice because he felt I should aim as high as I possibly could. Manhattan College had a very traditional engineering program, geared for the practicing engineer who would need a state license to deal with traditional electronic circuits, amplifiers, transmitters, receivers, filters, motors, transformers, and power management. Their electrical engineering curriculum had no computers in it. I graduated as a standard electrical engineer.

Yost: There wasn't a computer side of that?

Denning: The closest I got to any formal education about computers was their course on transistors. Computers were moving into transistorized circuits at the time. But even the transistor course had no computer switching circuits in it. In the summer between junior and senior years, I was fortunate to get a summer job with Bell Labs. My work there was closer to computers, but not in computers. They were developing a speakerphone at the time and needed a circuit to switch either end into send or receive mode. They had a circuit called "variolosser" that did this. My job was to take measurements of variolosser circuits and help tune them so that humans could not hear the transitions when they switched. That was very interesting but still not full blown computer switching circuits. The next summer, just after graduation from Manhattan, I worked with IBM in Poughkeepsie. There I really got into some of authentic digital systems. We worked on a large, complex switching device that connected disk units to the CPU. I was constantly inside the cabinets, checking that thousands of wires were properly secured to their pins on the back board. I invented a little device that plugged into the back board and indicated when the circuit was properly wired up. My manager liked it and filed an invention disclosure. It felt pretty good that I had done something that summer that might have been patentable. So that's about as close as I got computers during my time at Manhattan College. I finally did get to M.I.T. for graduate school.

Yost: When you decided to go to M.I.T. did you know that you wanted to focus on computers?

Denning: Absolutely. It was amusing that when I proposed to do what my father wanted all along, my father was happy, but most of the EE faculty at Manhattan were not. They told me that no Manhattan graduate had yet made it past the M.I.T. Ph.D. qualifiers. The faculty told me that M.I.T. teaches EE in a completely different way and that I was unprepared for the "M.I.T. way." I discovered one voice who had confidence in me: the dean of engineering. He told me I should follow my heart and go to M.I.T. I would find it very tough, he thought, but he knew I could do it. Well, that was what I really wanted, and I decided to go. I have been grateful to that man my whole career for his faith in me.

Yost: So initially, you applied for a master's and then …. ?

Denning: Yes, my immediate objective was a master's degree. However, on arrival, I immediately set out to find what would be the best strategy for passing the qualifying exam by the end of the first year. The first thing I learned is that I really needed to know the M.I.T. core courses. On reading the course descriptions, I discovered that five contained material I had never seen before. So I added them to my schedule. My schedule wound up covering what I needed for the masters degree plus what I needed for qualifying exam preparation – quite a load. In my first year I not only completed a master's degree but I relearned all of my undergraduate electrical engineering from the M.I.T. point of view. That was quite an experience, a second way of looking at the universe. The Manhattan view was very pragmatic, for example, What values of resistors and capacitors make the circuit work? The M.I.T. view was heavily principle-oriented; for example, what is the principle that makes the circuit work? I spent a lot of time relearning my EE from a principles-oriented view. Going back to strategy, I also learned that I needed to cultivate faculty who would speak up on my behalf in the faculty meetings where they make decisions. I asked Jack Dennis to be my thesis advisor and worked hard to complete a thesis he admired. When qualifying exam time came, I failed the first try and passed the second. Jack Dennis was a key voice in the committee's positive decision to admit me to the Ph.D. program. I had defied my Manhattan advisors who said I couldn't do it and justified the dean's confidence that I could.

Yost: In having the aspiration to complete a Ph.D. did you have ideas on what you wanted for a career? Were you thinking you wanted to be an academic?

Denning: I knew I wanted to teach. That went all the way back to my days in the high school science club. Ralph Money gave extra projects to his guys to help them develop their talents. In my case, he asked me to present lectures to the science club about what I was learning about electricity. So I developed a series of three or four lectures about basic electricity, with carefully written handouts. It went over very well. My club-mates learned their basic electricity and I was energized about my possible calling as a teacher. Ralph Money showed me that my life purpose is teaching. When I got into graduate aspired for the Ph.D., I already knew that eventually I wanted to be a teacher.

Yost: Did you have an opportunity to use the compatible time-sharing system (CTSS) while you were a graduate student at M.I.T.?

Denning: Yes, that the main system I used there. CTSS was just being put into production in 1964 when I arrived. I was one of the first student users. Jack Dennis had built a very experimental time sharing system on a PDP-1 computer and he had me writing programs for that system too. The summer before I had my first contact at IBM with a high level language, FORTRAN, but I did not write any large programs with it. CTSS introduced me to the MAD language, for Michigan Algorithm Decoder, which was a derivative of the Algol 60 language. I learned a lot of programming in MAD. I wrote a disk system simulator in MAD for my masters thesis. I wrote an I/O system in MAD for other users of CTSS; my program was much simpler and faster than the I/O package in CTSS. I learned a lot of programming with MAD. I also made friends with Al Scherr, who was completing his Ph.D. thesis on a performance prediction model for CTSS. He found that the classical machine repairman queueing model accurately predicted the throughput and response time of CTSS. It was quite a surprise to many people that such a simple model could be so accurate. To validate his model, Scherr did a lot of kernel programming to collect data. He showed me how to program well in the assembly language of the IBM 7094, the machine on which CTSS ran. Under his guidance I had become a full-fledged systems programmer. He taught me how communication between the user space and the kernel space worked in CTSS, how to read the kernel source code, how to implant data hooks, and how IBM had slightly modified the CPU instruction set to meet the M.I.T. requirement. He taught me efficient programming tricks for efficient assembler language programs, techniques not recorded into the manuals. He also taught me how the MAD compiler translated MAD source code to IBM assembly language. I was always grateful to Al for being my mentor and really helping me get through my master's thesis project, which was a simulation of the disk storage system. My interest in disk storage, kindled in my summer at IBM, blended with Jack Dennis's interest – can optimizing disk access times improve the performance of time-sharing systems? This question was also of interest in Multics.

Yost: And was that implemented to make CTSS more efficient?

Denning: I'm not sure. CTSS was a production system in my time and most of the research was focused on Multics. My simulations had demonstrated three things of potential use in CTSS and Multics. First, we proved that shortest-seek-time-first scheduling of disk requests is optimal. Second, we found that by clustering data on the disk, a form of locality, we could minimize seek times. Third, we found a new policy, scan, that moved the heads back and forth across the disk cylinders, reversing direction when there were no more disk requests ahead of it. The scan policy would pick up the shortest seek time next request in the current direction, even if a new request with shorter seek time arrived behind the moving heads. With scan, we could double the throughput of the disk. I know that some of these bits of knowledge found their way into disk controllers from IBM and others.

Yost: How did that feed into Multics?

Denning: When passed my PhD qualifiers I moved from being a student user of CTSS to a research assistant in Project MAC helping with Multics. I wasn’t writing code for Multics, but I interacted frequently with the Multics designers on disk-scheduling algorithms, the virtual memory, thrashing control, and the protection system. Although my PhD thesis was about resource allocation in large multiprogrammed computing systems, like Multics, my growing interest in protection systems was no accident. I was a big fan of virtual memory and I saw that many of the methods for protection used the same principles as virtual memory, notably the restriction of access to the pages or segments listed in a process’s page or segment table. Protection systems were basically using the same technologies we had already invented for virtual memory. It was an easy step for me to go to the architecture of virtual memory to the architecture of protection systems. Jack Dennis had worked extensively on this question with some of his other students.

Yost: Were you aware of any concerns regarding privacy or protection of data with CTSS back then?

Denning: Oh yes. I remember an incident where we had to change our passwords because a system operator left a copy of the password file printout on the printer, and someone filched it. There was another incident where a temp file left over from an administrator updating a password was still in the temp directory hours later and someone pinched it from there. The security of individual files was also an issue. Like every other user, I got a single, small directory for my files. There was initially no hierarchical structure, other than the system master directory listing all users with pointers to their individual directories. No one had access to anything in any other directory. The only way to share was to copy a file into the public directory, but then everyone could see it. A constant question on my mind was, "Should I trust the system to keep my files private?"

The Multics file system overcame these limitations. It was a fully hierarchical system that allowed users to create subdirectories at any time, and attach individual detailed access control lists to each file and directory. Multics was one of the first systems to use pathnames in the directory hierarchy as global names in the system. I could then reach out to any file or directory and read it if the owner allowed it in the access control. This idea of a shared file system with protection controls was critical to the Multics objective of sharing information in a community.

The idea of community helped us all be conscious that strengthening the technology was not the total answer to the protection problem. How do we get responsible behavior from users? System administrators? We need some form of education or training on good security procedures.

The protection question also reached outside of the immediate system. Somebody in CTSS invented the one-way password, where the login program would pass the user's typed password through a one-way function, and compare with the one-way code of the actual password in the master password file. This was a technological approach to protecting the password file. Now if someone did acquire the master file, they would have no way to reverse the encodings of passwords to discover the true passwords. A few years later, Bob Morris and others at Bell Labs discovered that this protection was not very good because users do not generate passwords randomly – they tend to use easily-guessed string like the string "password" or their own name. The found that in a community of 100 users it was virtually certain someone used an easily guessed password. Now we are back to a nontechnological question: How might we convince users to select passwords that cannot be guessed?

So I became very sensitive, almost paranoid, to seeing ways that data protection could fail and my files lost or stolen. By 1968, I could see a rising level of general concern about protecting the data and about preventing system penetrations.

Yost: Of M.I.T. faculty members, who were most involved and concerned with the protection system?

Denning: Quite a few. A major design objective of Multics was to provide a "computer utility environment." The system was accessible to a lot of people, it was highly interactive, it facilitated the sharing of objects, files, programs, and it provided strict access control. Everybody I ever talked to totally bought into these objectives. My advisor, Jack Dennis, had worked closely with Fernando Corbató, Bob Daley, and Peter Neumann on the structure of the file system. With his student Earl Van Horn, Jack wote a classic paper "Programming semantics for multiprogrammed computations" [CACM 9:3 (1977), 143-155], in which they discussed how to organize an operating system that fully protected every object in the system. That paper became the basis of architecture for many future operating systems and for object-oriented run-time systems. While not part of Multics, their work was grounded in their experience in Multics, and it anticipated and solved protection problems that were discovered in Multics and other systems. Jerry Saltzer was another influential faculty member; he was a designer of CTSS and major player on the Multics design team, and was a student of Fernando Corbató, as was I. Still another was Bob Fano, who was head of Project MAC and was a major figure in communication theory. The leadership of Project MAC had all made protection and security a priority of the Multics system.

Yost: I understand your dissertation committee included basically all these individuals — Saltzer, Fano, Corbató — quite a stellar group.

Denning: It was a good group, yes.

Yost: Can you talk about how you came to your dissertation topic and briefly summarize it?

Denning: The topic that I chose was kind of a natural extension of my master's thesis. My master's thesis, as you recall, was about how the scheduling of disks to minimize the swap time. Multics was much more complex; disks were just one of many resources that Multics had to manage and schedule. The Multics team was looking at two sides of sharing, which I call architecture and allocation. Architecture is the structures of the operating system that allow the various resources to be accessed. Allocation is the policies that decide which users get which resources at any moment of time. Students who studied architectures for sharing and protecting resources were Roger Schell, Mike Schroeder, and Earl Van Horn. With Allan Scherr, I was a member of a small group who studied policies and control systems for allocating resources, and overall system performance.

These two sides of Multics – architecture and allocation – reflected two deeper concerns that all systems designers face. One is correctness, leading to architectures that enhance correctness. The other is performance, the ability to predict how the system will do with respect to key measures such as throughput and response time, control the system so that it operates in acceptable ranges, and design the system to have the capacity to meet its performance targets and avoid bottlenecks. Without both correctness and performance, the system will be useless.

My PhD project dwelt on the performance side. I had already been initiated into this side through my masters thesis and my work with Al Scherr. So I was ready to go on the performance side. I was drawn to memory management because I found the idea of virtual memory to be wonderfully elegant and I knew that there were serious performance issues before it would be viable. I had many conversations with Jerry Saltzer about the

Multics virtual memory. I also had many conversations with Professor Richard Kain, who was a skeptic of virtual memory but exceedingly helpful in focusing me on the questions that needed answers to win over skeptics like himself. After I graduated from MIT, Kain moved to Minnesota and had a long career as a computer scientist there. Is he still there?

Yost: I don't think he is on the active teaching faculty, he might be emeritus. That probably came up in your interview with Arthur Norberg—I seem to remember that.

Denning: Yes. Multics virtual memory was a big deal. The designers were building in all the latest ideas, such as pages, segments, protection rings, system-wide name space, page replacement, and multiprogramming. Multics was the first system where all these ideas came together there wasn't a lot of prior experience to go on. They were concerned and cautious. I do believe that the early Burroughs systems used segmentation in their virtual memory. Burroughs systems were attuned to the ALGOL compiler, which allocated objects such as arrays and procedure codes in their own memory segments.

Burroughs's experience was good but didn't scale into Multics. Jack Dennis wanted to define the virtual memory to include the entire space of files and directories. The pathnames in the directory hierarchy solved the difficult problem of how to name files uses wanted to share. Up to that time, virtual memory and file systems were separate concepts. In Multics, you didn't have to open a file because it was already in your virtual memory. But this created new challenges for the system designers. For example, how to automatically link a previously unopened file to a computation that just referenced it for the first time?

Anyway, when I signed on to Project MAC and Multics in 1965, the performance aspect, especially the virtual memory, was high on their minds. They had pretty much settled on the architectural approaches and the question was, will it deliver the throughput and response time we want? At the time, page replacement was a big unknown. The preliminary evidence from other virtual memory projects was that page replacement was a critical performance factor. Get it wrong, and the virtual memory is useless. There were many proposals for how to chose the page to replace from a virtual memory on a page fault. The objective of page replacement was to minimize page faults in the virtual memory. Should it be the complicated "learning algorithm" used on the Atlas machine in 1959? Should it be first-in-first-out (FIFO), which was incredibly easy to implement? Should it be least-recently-used (LRU)? Something else? Each proposal had its strengths and yet it was easy to find examples of programs that performed poorly under that proposal. Regardless of which policy was in effect, how come the load order of the subroutines had such a tremendous effect, good or bad, on performance? At the time, page replacement was finicky and poorly understood. The idea of a virtual memory, attractive in principle, looked shaky in practice. The skeptics, like Richard Kain, saw too many conflicts in experimental data, too many differences among programs, and too many unanswered questions about interactions among different programs competing for the same limited memory space.

You know, that load order issue was a real challenge. In the old systems, programmers would stack up their cards for each subroutine into a deck and submit the full deck to the system via a card reader – the order of subroutines in the deck was called "load order." Later this was automated by "linking loader" programs, but the same issue arose because the linkage order affected the positions of subroutines in the address space. This turned out to be important because of paging. It was easy for two subroutines that were called near one another in time to wind up on different pages, and each subroutine call caused a page fault. If those two subroutines were located on the same page, only one page fault would be needed to get them started, and after that they could call each other without causing more page faults. It was very difficult to tell now to measure "affinity" between subroutines so that high-affinity routines would be assigned to the same page. The known algorithms for doing the clustering had huge overheads even for a relatively small number of subroutines. Later, the complexity theorists told us that this problem, known in mathematics as "bin packing" was NP-complete. No wonder we could not find an efficiently solution for computing good load orders!

Saltzer told me that if I could help them understand how to make page replacement be robust and efficient despite all the finicky-ness, that would be a major contribution to Multics. He told me about another problem they had encountered, and they did not understand. He called it thrashing. It was an unanticipated problem that appeared when they used multiprogramming and virtual memory together. They observed that when the level of multiprogramming (number of programs loaded into main memory) exceeded a critical and unpredictable threshold, the system throughput would suddenly collapse. The programs were still running but were stuck in state of constant paging and could make almost no progress. The engineers said that the system was "paging itself to death." Apparently other vendors such as RCA and maybe IBM had encountered the same problem. If they could not find a way to stabilize against thrashing, virtual memory systems would be a multimillion dollar liability. Saltzer told me his dream. He said they would like a memory management policy that was simple and had just one tunable control parameter. We would then adjust that parameter for maximal throughput and leave it at that setting. Internal control systems would automatically adjust the multiprogramming level to avoid thrashing. Figuring out how to solve "Saltzer's Problem" became the main challenge of my PhD thesis work.

Yost: In focusing on these challenges that are under the broad category of efficiency in operation, were you at that time developing any kind of an interest in the other side, the correctness and protection aspect of that?

Denning: Oh, yes. Jack Dennis was primarily interested in architectures that supported safe and reliable computation. I had many conversations with him and his students, my office mates, on such matters. Multics was making a big promise: a general purpose programming system that allowed anyone to share anything and always respected the access rules declared by every object's owner. How could any computing system be organized to deliver such a promise?

Jack thought the solution of this problem depended on finding various invariant properties that would support "protected sharing" and constructing the architecture to make sure those properties held, no matter what. The "no matter what" had to include machine errors and intentional attacks as well as program bugs. Then, with the help of the architecture, a small set of essential protected-sharing properties would produce a highly secure system. I don't think they had much interest in some sort of mathematical proof to show that the system had all the properties because it was too hard. The only approach was inherently empirical: build and test prototypes on realistic conditions.

One of the reasons Jack and others persisted with virtual memory was that its built in address mapping mechanism automatically partitioned the memory into protected regions. An individual program could refer only to its own pages and had no means to refer to the pages of any other program. Virtual memory's logical partitioning property was a key element of a protected system. It really does give excellent enforcement of an isolation principle when built into operating systems. In short, the Multics designers needed the virtual memory to achieve the Multics promise.

Yost: In the time that you were in graduate school and in the work on Multics, was there a sense of the people who worked on it, the team, of the issue with government classified records? In 1967, Ware and Peters presented a pioneering paper on the problem of multi-level computer security in time-shared environments. Was that at all influential?

Denning: I was not part of any discussions about multilevel secure operating systems that might have taken place at Multics. I came across those discussions among people designing other operating systems. I remember Roger Schell working on his PhD thesis at the same time I was there. Roger's thesis was about the design of a kernel that would enable the automatic reconfiguration of modules, hardware and software, making up a computer system. It was not about multilevel security, but about how to change the system on the fly without causing errors. Roger's experience with kernel design served him well some years later when, in his Gemini project, he designed and built an operating system that met the Orange Book A1 requirements. I am certain that in his design at M.I.T., he was keenly interested in the correctness in the kernel to protect against all sorts of threats -- how to organize the kernel to give you the maximum assurance that it was doing the right thing. He got pretty deep into that. I remember he had a list of seven different levels of sharing you could have in a system. They were complete isolation, sharing copies, sharing originals, sharing services, handling mutually suspicious subsystems, providing memoryless systems, and providing fully certified systems. I think Jerry Saltzer and Michael Schroeder helped with that list. I worked on some of those myself a few years later when I was at Princeton and Purdue. I designed capability architectures for mutually suspicious subsystems and worked a little on memoryless systems that erase all their data on completing their function. Roger himself was interested in the hardest problem, certified systems. He worked on that in his PhD thesis and later started a company (Gemini) to build a provably secure operating system.

When you get to the higher levels of sharing, you start to get deeply into issues of trust. We try to find technical methods to support trust, but we cannot completely eliminate the need to trust other human beings. For example, I might find a technical way to keep my accountant's software from retaining any information about me and my finances; but I still have to trust that my accountant won't divulge or exploit information we discussed in our conversations. At some point, I have to trust that you actually do what you said you were going to do.

Yost: As a graduate student, Roger Schell, coming in from the Air Force, was really influential to this?

Denning: Yes. Although he did not have a formal background in computer science, he was quite familiar with military security requirements and a clear thinker to boot. He was able to zero in on the really important issues that have to be solved to make a system reliable and secure. Over the years, he made many contributions to cyber security, and was named as one of the original eleven members of the Cyber Security Hall of Fame in 2010.

Another extremely influential person was Jerry Saltzer, who was Roger's advisor. Jerry also worked close with another student, Michael Schroeder, on the design of the Multics ring protection structure, which was a way of implementing a hierarchy of supervisor states. Every program was assigned to a ring, and lower number rings had more privileges than higher numbered rings. A program could call another in a higher numbered ring, but had to use a protected entry point for a call to a lower numbered ring. The protected call set the callers ring to the called ring and restored it on return.

The ring design had to deal with a tricky problem involving parameter passing. I might want to write a value into a file for which I had no privilege. I could pass the file as a parameter to a procedure with more privilege than me, and it could write the file for me. This could get even trickier if that file parameter was passed along on more calls to other procedures. Mike found a way to determine the highest privilege ring through which such a sequence of calls might pass, and allowed the final read or write only if the final procedure had enough privilege to write that highest ring. The final design was a very sophisticated system for managing the flow of control through all the kernel without, at any point, having any risk of the higher privileged entities on a path doing something that was not allowed to the lower privileged entity. It was extremely clever and very well done.

Yost: You graduated in 1968?

Denning: That's right.

Yost: And your first job out was at Princeton?

Denning: That's right.

Yost: Can you discuss both that environment and your research interests during the Princeton years?

Denning: Let me say something about the decision to go to Princeton. As I was finishing up at M.I.T., various other schools invited me to apply for faculty positions. I was in operating systems, a fairly new field, and a lot of departments were looking for people in that area. I talked to M.I.T. people about getting an appointment at M.I.T. They made me an offer, but they also wished I would go to another department for a while and then come back to M.I.T. The salaries in my 1968 offers would astound you by today's standards. M.I.T. offered $9,000, Princeton $10,000, and Cornell $12,000. I think

Stanford also made an offer in the same ballpark. Today's most junior teaching assistants are paid more than twice those amounts!

Yost: My father started as an academic at that time so I've heard about those numbers—actually a bit lower as he was a historian also.

Denning: My family, who were New Yorkers, strongly favored Princeton and were not interested in remote places like upper New York state and California. I remember one day as I was struggling with what choice to make, Lotfi Zadeh visited M.I.T. and we had a long talk about careers and futures. We hit it off. His advice was go to a different place from where I was brought up (M.I.T.); learn their ways and become good at them. He thought I would do well away from M.I.T. because he saw me as an independent operator who can make a place in any environment. He thought it would also be helpful to make a name for myself independent of M.I.T. That conversation was a big influence. My family concurred that Princeton would be an ideal next place for me after M.I.T. and had equal name recognition. It was difficult to make the $10,000 salary work, but with a summer job I was able to do it.

Yost: Right.

Denning: I moved to Princeton in summer 1968. I was in the Electrical Engineering (EE) department. The department had a small and very strong computer science group. But the CS group was not the main power center of the department. The power center was the larger, more traditional EE group on signals and systems. There were a lot of politics between the two subgroups.

Yost: Roughly how large was this computer science subgroup?

Denning: I don't remember the numbers anymore. The whole department might've been in the mid-teens; two-thirds of them were the EE system folks and one-third of us were the CS folks. It was really a EECS department with the main emphasis on EE.

After arriving at Princeton, I found out that there were campus-wide forces trying to influence the direction of computing at Princeton. Bigger than in the EE department. One of our distinguished faculty was John Tukey, a well known statistician and numerical method expert. He was a big fan of high-end computing. At the same time, Ed McCluskey of the EE department was a big fan of interactive computing. They had been arguing over whether Princeton should invest in a supercomputer or a time sharing system. McCluskey left for Stanford about the time I arrived, but the battle over computing direction went on.

From my perspective, all the politics were about money. A time sharing system benefited EE more than a supercomputer, so naturally we favored time sharing. In addition to that, when new money came to the department, say for a new faculty position, we would argue over whether it goes to the systems group or the CS group. Similarly, if a promotion is allowed to be in the department, should it be for the systems candidate or the CS candidate? I was just an assistant professor so I kept my head down and didn't get involved in those issues. I just collaborated with colleagues in the CS group. My most active collaboration was with Ed Coffman -- we wrote a book together. I also worked a lot with Jeff Ullman and Al Aho and we published a couple of papers together. Aho visited frequently from Bell Labs. I collaborated with Bob Keller. Occasionally Don Knuth and Brian Kernighan visited the department and we collaborated with them. I also had some very good students. Scott Graham completed a masters thesis on protection and security and later a PhD thesis with me at Purdue on program behavior modeling. Jeff Spirn did a PhD thesis on program behavior modeling. It was a rich and rewarding intellectual atmosphere.

Yost: Had Graham published an article back in 1968 in *Communications of the ACM*?

Denning: Which article is that?

Yost: "Protection and Information Processing Utility" in *Communications of the ACM*.

Denning: Oh, that would be Robert Graham, who was part of Multics at the time and then went to University of Massachusetts.

Yost: Okay, that makes sense. It didn't make sense that someone at the master's level would be publishing in *Communications*.

Denning: Scott Graham was just a graduate student. Bob Graham was a Multics collaborator.

Yost: Can you elaborate on what research you were most interested in at the time?

Denning: I brought my M.I.T. interests with me to Princeton, mainly in the memory management and protection areas, and I developed a new interest in operating system theory in collaboration with Ed Coffman. With Scott Graham, I continued the architectural work on protection and security. We wrote a paper in 1972 called "Protection Principles and Practice." It was the joint work from Scott's masters thesis, basically. We discussed what it meant to share and protect, what was available in the architectures, what was available in the models. The most advanced protection model at the time was the Lampson model of the access matrix. We built on that.

Yost: Can you discuss the significance of that model?

Denning: Sure. It has a bit of interesting history. I actually had something to do with it. In 1969, Bruce Arden asked me if I would serve on a project that he was leading. At least I think it was it Bruce Arden. It was NSF-funded project called COSINE, meaning Computer Science in Engineering. They were interested in specifying core courses in computer science that could be included in electrical engineering programs. They wanted a core course on operating systems. Bruce asked me to chair a task force to do that. I put together a small committee and COSINE provided a travel budget. The committee was Jack Dennis, Butler Lampson, Nico Habermann, Dennis Tsichritzis, and me.

Each of them brought a special strength to the committee. Dennis was a supremo architect and a champion of operating systems principles. Lampson was the chief designer of a time sharing system at Berkeley and had numerous ideas about data protection. Habermann was a student of Dijkstra and had helped design the THE system as well as some novel deadlock avoidance algorithms. Tsichritzis was an expert in data management. Because of the huge positive response to the ACM symposium on operating systems principles, which Dennis organized in 1967, we were certain there was a wealth of principles to make a core course. We divided OS principles into a few groups including process management, memory management, naming, protection, and design. Then we divided up the work and each member drafted recommendations in one of the areas.

Yost: Who worked on protection?

Denning: That was Lampson's, he was really interested in that one. At one of our working meetings he said he had figured out how to present the basic ideas of protection. He presented a model with a large number of sets, one for each pair of process and object in the system. The math notation was so dense, I don't think any of us was following him. Dennis Tsichritzis went at him. He said, "Butler, I respect you a lot but I don't know what the heck you're talking about! Can you draw me a mental picture?" Butler offered more mathematics and Dennis finally said something like, "It would help if you drew a matrix?" and he went to the board and sketched out an example. Dennis said, "Is this what you're talking about?" And Lampson replied, "Yes that's what I'm talking about." And Jack asked, "Could you present it this way? Then I could get what you're talking about. I'm a pretty experienced guy and if you can't communicate it to me, how are you going to communicate it to students?" So Lampson said something like "Arrrrr, mumble." But for the next working meeting he had it reformulated with a matrix. It was beautiful. The matrix had "domains" down the side and "objects" across the top. A particular entry for a domain and object was a list of permissions for the object. He showed how to derive normal concepts like access control lists and capability lists from the one matrix. Basically, an access control list reads the matrix down a column, creating a list of permissions that each domain has to the object. A capability list reads the matrix by rows, enumerating the permissions a particular project has. A lock-and-key system selects groups of processes and objects. This model would obviously communicate easily to students. We could actually explain what's going on in different operating systems, based on the matrix. We thought that the matrix model was nifty. That's how we helped Butler. Then he wrote it up and it became a classic paper.

Coming back to the task force, our report incorporated Lampson's access matrix model as one of the main elements. The final version of the report was finished in 1970 and distributed in 1971. Our model of an operating systems course envisions modules on processes, memory, naming, protection, and design. Every one of those main topics is still a main topic of operating systems today. We got it right and it has stood the test of time. Many authors incorporated the outline into their OS textbooks. We turned the tide and operating systems was the first system-oriented core course adopted into the core curriculum of computer science. It is still there.

I followed up with a *Computing Surveys* paper in 1971, "Third Generation Computer Systems" in which I described all the architectural structures built into the current generation of computing systems.   I think that paper helped a lot of teachers understand the conclusions of our COSINE report.

Yost: With these two articles you had really established yourself in computer security.

Denning: Yes, I had become a voice in the security area as well as other operating systems and computer systems areas. Not long after the COSINE report was distributed, Ed Coffman and I decided to work on the book called *Operating System Principles*, focused on the performance side of operating systems, not on the architectural side. That became a classic book, too. But it wasn't about protection security. There was nothing in our book about that.

Yost: Yes, I went through that textbook to see if there was anything. Did you ever consider including anything on security?

Denning: No, Coffman and I decided on a performance perspective. Everything in the book was about how processes, schedulers, memory systems, and computer networks behaved. The only theory we were aware of about protection was about completeness and consistency of protection policies described by access matrices – applications of the Gödel theorem. That was outside our scope.

Yost: In the 1971-72 period you investigated potentially moving to other universities. Can you discuss that?

Denning: Yes. In my fourth year at Princeton I became interested in my promotability at Princeton. Princeton was really tight about promoting. The university had a cap on the number of tenured faculty it could have. They were at their cap. Only when tenured faculty retired were there new openings for faculty to be promoted.

[RECESS]

Denning: My department chair estimated that it was likely to be five years before the engineering school was authorized to make a promotion. It could be sooner if there were more retirements than expected. On top of that, there are five or six departments in the Engineering School with candidates for promotion. What is the chance of one of those slots coming to the EE department? And then what is the chance it will make it to the CS subgroup, which had only one-third of the votes? This was not an encouraging environment.

Yost: Sounds like considerable risk.

Denning: Remember, I was only in my year four at Princeton. Most promotion cases are brought in year six. My chair told me that if I had any notion about going up for early tenure, I should forget it. It would not be considered in this environment. I was not optimistic about my prospects.

Not long after that I encountered Sam Conte at a conference. Sam was head of the CS department at Purdue. As we rode an elevator together, he said, “I hear you’re not happy with your prospects at Princeton. What would you think of talking to us at Purdue?” He said some things about what makes Purdue great. By the time we got off the elevator I said, “Yes, I’m interested in talking.” Talk about a successful elevator pitch!

I soon went to Purdue in the dead of winter and liked the place and the faculty. Although they had a strong reputation in numerical methods, they also had a strong systems group that included Saul Rosen and Herb Schwetman. Sam wasted no time in getting me an offer letter. He offered me immediate tenure at the rank of associate professor and a 50 percent raise of salary. That was one of those dream offers you can’t turn down.

Yost: Purdue was the first or one of the first two computer science departments, is that correct?

Denning: Yes it was. Both Purdue and Stanford founded computer science departments in 1962, and I think Purdue made their announcement earlier in the year than Stanford. Technically Purdue was the first.

Yost: In addition to it being a very attractive offer, was that another plus in your mind that it was the first and its own department not a subgroup within EE?

Denning: It was a strong department and it was a good university. Obviously I wouldn't have to worry about tenure any more. They wanted to build up operating system area, teach courses like the one I designed at Princeton, and design new courses. As I said, it was like a professional dream come true. With my EE background, I quickly got to know the EE faculty and collaborated with a few. I also enjoyed learning about the science disciplines in my school of science, and started thinking about computer science as a science, not a misnamed engineering field.

Yost: In your paper, "Protection Principles and Practice," you cite a NASA grant, I assume you rather than Graham was the investigator.

Denning: Yes. As I recall, while at Princeton I got a couple of exploration grants from NASA, the space agency, issued through their Langley Center in Maryland. The work was done at Princeton but some of the publications from them came when I was a Purdue. Scott Graham was a master student at Princeton, where he worked on his thesis on protection. He came with me to Purdue so that he could continue on to a PhD program in operating systems.

Yost: NASA was interested in protection?

Denning: Yes, NASA gathered large amounts of satellite data that had to be protected. They were interested in possible system structures that would reliably protect data. When Scott finished his thesis, we wrote a paper about his main findings for the 1972 Spring Joint Computer Conference. One aspect of that paper stuck in people's minds – the notion we called reference monitor.

The reference monitor was the notion that every class of objects had a system that managed it, and that system enforced all the access rules registered in the access matrix. For example the process manager, virtual memory manager, file system, and directory managers were all reference monitors for processes, semaphores, virtual address spaces, files, and directories. When one of your programs attempts access to any object, the reference monitor checks that you have permission. For example, if I log in and run a program "foo", the process running "foo" is tagged with my login identifier "pjd", showing me as the owner of the process. When "foo" attempts to read a file, the file system checks that the owner "pjd" has read permission. If "pjd" does not have read permission, the file system generates an error signal. The principle that a reference monitor validates every access is necessary for the successful implementation of a security policy representable as an access matrix. It's a simple idea, really.

Yost: This originated the concept of the reference monitor.

Denning: Yes. Other people picked it up. James Anderson, a well-known security consultant, promoted it in his community, saying that the biggest contribution of that paper was the reference monitor. That became the standard notion in everything he talked about when he was talking about how to make a system more secure; the system needs a reference monitor for each kind of object in the system.

Yost: I interviewed Roger Schell and that came across strongly in that.

Denning: Right. I'm sure Roger already figured out the necessity of the principle before Graham and I wrote about it. To me, at the time when we wrote the paper, it was like an obvious thing. We needed a precise way to ensure accesses were being checked so that the structures later in the paper would work. It seemed to me that we were just recording common wisdom. Maybe we were the first to put a name on it. Process managers, virtual memory managers, and file managers in operating systems all worked this way. Lampson assumed that the system implementing an access matrix worked this way. Graham and I did not think of reference monitor as the main contribution of the paper; at the time, we thought we had a good solution to the problem of mutually suspicious subsystems interacting with each other.

Yost: You mentioned four systems that embraced concepts that you identified in your model. They included Fabry and Wilkes. Can you go through those two and comment?

Denning: I think you are referring to Robert Fabry's work on capability addressing and a prototype capability machine called MAGNUM he designed for the purpose, and to Maurice Wilkes's work on the Cambridge CAP capability machine. They are two approaches to the design of a system based on capability addressing.

The idea of capability addressing came from the Dennis and Van Horn paper in 1966 ("Programming semantics for multiprogrammed computations"). Fabry took the key idea from that paper and turned it into his Ph.D. thesis. He came up with a novel way of interpreting a capability as a unique identifier for an object. A capability acted like a ticket: Anyone who had a capability for an object could access the object with no questions asked by the system. You could only access an object if you had a capability for it, for example, issued to you by the owner. Fabry said that any large random pattern of bits would work as a capability as long as it was unique. He thought of a capability as "a magic number that conferred access to an object", hence the acronym MAGNUM for his machine. A key feature of a magic number was that the machine would not allow it to be modified after it was created. In his thesis he laid out an architecture for a machine that used capability addressing instead of the familiar location-based addressing. MAGNUM was the first attempt to define an architecture for the principles of the Dennis and Van Horn paper. In 1974, Fabry wrote a paper "Capability based addressing" (ACM *Communications*, July 1974), where he showed that capabilities were the only viable method to implement global sharing -- and protection -- of objects in a large system.

Maurice Wilkes, head of the computing laboratory at University of Cambridge, was always on the prowl for new ideas. He took an interest in Fabry's work and noted that such a machine would present challenges in the design of its hardware and its operating system. Around 1974, with David Wheeler and Roger Needham, he started the CAP project at Cambridge to build a working capability machine and operating system. They wanted to put together, in one machine, all the new ideas about capabilities advocated by Fabry, Dennis, and Lampson, and see if the result system would deliver the high degree of sharing and protection promised. At the end of their adventure, they wrote a book about their system, *The Cambridge CAP Machine and Its Operating System* (Elsevier 1979). They said the system achieved many of its objectives. It was hard for errors to propagate since processes only had a limited set of capabilities each, and that did not create very many channels for error propagation. It was great for debugging because most errors quickly causes some sort of capability violation, causing the machine to stop dead in its tracks. It was a huge advance for software reliability and debugging.

Yet they came to a pessimistic conclusion. They said that the operating system was too complex to be viable for a commercial environment. The reason boiled down to their fundamental principle of capabilities – that once a capability was created, it was impossible for any machine operation to modify it. They wound up having to make two copies of many operating system modules, one to handle the flow of capability information and the other to handle the flow of normal data. It seemed like two parallel operating systems were cohabitating in the same machine.

I was never able to reconcile Wilkes-Needham conclusion with the experience of the Plessey Company, which built a capability machine for message switching. They had a very positive experience and were strong advocates of Fabry's idea. Perhaps they succeeded because message-switching computers did not need a general purpose operating system.

Some years later, object-oriented programming emerged into the world. The implementation of objects used "handles" as bit patterns that pointed to objects, and functioning in exactly the same way as Dennis-Van-Horn capabilities. Handles were not subject to an absolute requirement that no one could tamper with them. The object run time systems hid handles from the users, so that no user could tamper with them, but treated them as pointers within software designed not to tamper with them. That was a valid way to implement the Dennis-Van-Horn idea and a relaxation from Fabry's insistence on a tamper-proof magic number. The capability idea is alive and well in today's object oriented systems.

In the early 1980s, the tide in the architecture world started to turn away from sophisticated structures like capabilities and toward very the very simple structure of RISC (reduced instruction set computer) chips. The simplicity of RISC structure facilitated the industry's ability to keep producing chips at the rates predicted by Moore's Law. RISC advocates argued that any function in a capability system could be implemented as a subroutine in a RISC system and probably run faster. So many of the capability addressing ideas disappeared from architecture. But, as I said, they survived in the software structures of object-oriented systems.

The RISC argument was powerful and persuasive, and won the day. Unfortunately, many of the things that used to be in the hardware to support operating systems disappeared and the security of software went down. There is a movement today, led by Peter Neumann of SRI, to see if computing in the Internet can be redesigned with a clean slate to achieve the needed levels of security, privacy, and protection. That project is likely to resurrect some of the concepts that disappeared in the RISC revolution.

In 1980, just as the RISC argument was gaining a following, Glen Myers, a very creative computer architect, published a book about an architecture he called SWARD, which was finely attuned to object oriented computing and had extraordinary reliability and protection properties. But the RISC revolutionaries cited SWARD is an example of "complex instruction sets run amok." That idea too got washed away in the RISC revolution. Have you heard of SWARD?

Yost: I don't think so.

Denning: It was an object-oriented machine, which directly represented and interpreted objects in memory. It stored an object as a descriptor along with its content, in effect embedding the metadata with the object. Most instructions accessed objects by selecting fields and subfields, rather than by giving memory addresses. This enabled dynamic type checking and access checking on each and every object at every access. By bringing the machine instructions closer to the user's object space, it reduced what Myers called the "semantic gap" and reduced errors in translating a programmer's intention to machine code. Myers claimed that narrowing the semantic gap between the machine and the user's language made the system less error prone and more secure. His book came out just around the time the RISC revolution was gathering its steam. SWARD was left in the dustbins of history. This really ingenious design is now totally forgotten.

Yost: Interesting.

Denning: A lot of the ideas about how to move from a very strict capability system (like Cambridge CAP) to a hybrid technology that would approximate capabilities with more standard hardware never really got explored. Researchers were trying to figure out how to minimize the instruction sets; how to build the compilers for the pipelines; how to build the chips; what functions to put into instructions versus subroutines; and so on. Many of the lessons from experience with capability machines for improving security and reliability got lost because of the way that technology moved. Today, a lot of security architects don't even know a lot of these things exist. Many say that any work over five years old must be obsolete. What a way to lose timeless principles!

Yost: It's one of the reasons we're doing this project.

Denning: Yes. One of my colleagues, Craig Martell, tries to explain this by examining almost independent lines of development of computing systems. The first line began in the 1940s and was very strong through the 1980s; it focused on the architecture of large computing systems that held a community's data and allowed any of it to be shared if the owners so wished. In the early 1980s, a new line was born, the PC or personal computer line. Its founders rebelled against the size, complexity, and cost of "mainframes" and offered the possibility that any individual could own and operate a computer. To do that, they worked with consumer electronics and inexpensive components. Their machines were very simple and not very powerful, but they were personal. Their designers often demonized mainframes, saying they were too complex and only affordable by rich corporations. They wanted to keep things really simple. Their hardware was simple, the size of data words in the machine small, the programming languages and operating systems really primitive.

Some of this bifurcation could be seen as early as 1970. The designers of Unix, Dennis Ritchie and Ken Thompson, wanted an operating system with most of the functionality of Multics that could run on a cheap minicomputer. Unix was the result. Unix started out as reactionary, tailored for the new world of minicomputers, but became part of the mainstream PC designers were trying to avoid in the 1980s.

In the PC world, virtual memory didn't exist until the late 1980s and even then it appeared in only a few operating systems like OS/2. Multitasking didn't exist until Windows and Apple Mac systems turned to it in the 1990s. Multicore chips and parallel programming did not exist until after 2000. One by one, as its computers get bigger and more networked, the PC world has been rediscovering the same problems encountered with mainframes thirty years before. By and large they are not aware of this and have tried to solve the same problems from scratch. When told that solutions were discovered thirty years before, they would react that those solutions don't apply in a PC world.

I recently met a delightful guy, Adrian McMenamin, who completed a masters thesis in 2010 on whether locality principles applied to Linux. He started out thinking that working sets and thrashing were relics of "Olden Times," but soon discovered from his experiments that locality was alive and well in Linux programs. If anything, it was more pronounced in modern programs than it was in olden programs. He showed how memory management in the Linux kernel could be improved using locality principles, and he gave up his skepticism. Many other PC world programmers are not as open to learning from the past as he was.

Anyway, there you see the two universes Craig was talking about: the ancient history mainframe computer system, and the modern mobile PC based networked system. The latter are having trouble solving problems that are the same as ones once encountered by the former.

Yost: What do you recall about IDA or TAL?

Denning: Sorry, I never worked with IDA or TAL systems.

Yost: What about the IBM RACF product? It's an access control product that was first tested as a concept on their customer base at the IBM 1974 SHARE meeting and by the mid-1970s it was their standard access control product, in fact, in a different iteration, it is still an IBM product.

Denning: I don't know that product either. But in looking it up in Wikipedia, I see that it stands for Resource Access Control Facility. When introduced in 1976, its objectives included user id and password checking, classification and protection of systems resources, access rights to protected resources, controlling the means of access to resources, and logging and auditing of accesses. Today it also supports digital signatures, public key infrastructure, directory lookup services, and case sensitive passwords. It rests on "zSeries" hardware to protect digital certificates with tamper-proof cryptographic processors, and it can support Multilevel Secure (MLS) Systems.

Yost: Let's switch topics. You met Dorothy Davis at Rochester in 1971?

Denning: That's right.

Yost: And she came to Purdue in 1972.

Denning: That's right.

Yost: Initially, she was your student. Did she come to work for you?

Denning: When I met her she was talking about getting a Ph.D. I told her that Purdue, where I was about to move, had some openings. I offered to talk to Sam Conte to introduce her. Sam was impressed. He said she could teach courses that she taught at Rochester and he needs taught at Purdue. He wanted to bring her in as an instructor. She took that offer and taught courses while enrolling into the Ph.D. program.

Yost: Was she interested in computer security before she came to Purdue or did she develop that interest there?

Denning: At Rochester, she worked with and taught systems programming and compilers. At Purdue, she became very interested in the architecture of secure systems. She did a project in an operating systems course around that topic. She went to Herb Schwetman to be her PhD advisor. He was particularly interested in maintaining a focus on the question "What's the system behind all of this?" That appealed to him as a systems guy, and to her as a former systems programmer. She credits him with continually asking that question and keeping her grounded.

I remember the time when she came up with the lattice model for secure information flow. She put together a mathematical description of how to compute a security class of information generated when two existing security classes are combined, and how to describe the most privileged security class the two existing could both read. Fortuitously, the famous mathematician Garrett Birkhoff was visiting John Rice that semester and had written a book on the subject of lattices. Herb suggested she talk to Birkhoff because he believed her mathematical model was a lattice, and he wanted her to take full advantage of that math from the world's leading expert. The environment of the department was wonderful because we had so many mathematicians available to help us sort through the mathematical models of systems of all kinds. Of course, this appealed to Dorothy, who was herself a math major at University of Michigan.

By that time we were married and I was concerned about getting in her way. I didn't want to compete with her in the security area, or generate professional conflicts. I decided to drop out of the security area and pursue the other areas where I was having ideas and student interest, including operating systems, architecture, and performance evaluation.

I did have one student, Mayer Schwartz, who completed his PhD thesis on security of databases – how one might ferret out supposedly secret information by asking a series of clever queries. He had signed on as my thesis student before my security declaration and I wanted to help him complete his work. Mayer moved to Oregon after graduation and worked for Tektronix and later for Intel.

I had another student, Don Dennis, whose interest partially overlapped into security. He was interested in how to design an architecture so support a secure kernel and capability addressing. He came up with some very clever designs. I remember one of them was implementing the WAIT and SIGNAL operations on semaphores. His method did each of those operations in two or three instruction cycles and did not require protected entry points as for normal OS software routines. Unfortunately, the RISC revolution underway at the time viewed his design as "complex instruction set" and it didn't go far. Years later, many operating systems designers wished they had fast instructions to so some of the primitive OS operations. Don moved to Oregon after graduation and had a long career with Intel.

Yost: Were you aware of the work that Bell-LaPadula published in the 1973-74 time frame and what was your assessment of it?

Denning: Yes, I was mildly acquainted with it but I didn't pay much attention to it because my interests were more with the architecture. I remember reading it and finding it hard to follow and mathematically over-complicated. I remember asking Dorothy about it and, when she re-explained it in terms of the much simpler lattice model, I got it. Still, their model made a big difference for the Air Force. It made quite an impression. It is still cited today.

Yost: The IEEE-CS, of course, started their annual symposium on security in 1980. Were there conferences before then that you attended? Maybe not annual ones but events that you went to that brought computer security people together?

Denning: I was deeply involved with SIGOPS, which was the initial focal point in ACM for security issues. In 1969 I organized the second SOSP, the symposium on operating system principles, and helped launch SOSP as a permanent feature of SIGOPS. SOSP's always seemed to have some papers on security issues. I vaguely remember a workshop, possibly hosted by SIGOPS, where designers of secure operating systems came together, but the most I can recall about date would be "sometime in the 1970s." I do not remember when others formed their own SIGs around security and started their own conferences on the subject. I remember taking part in a special month-long workshop at Woods Hole, MA, on the security subject some time in the early 1980s. I helped out with the CFP – computers, freedom, and privacy – conference in 1980 when a lot of people were getting interested in legal and social aspects as well as the operating system and network aspects. Our initial objective with CFP was simply to get the various communities together and talking in the same room. Dorothy was a lot more involved in those security conferences than I was.

Yost: In the first half of the seventies, was there a sense that there were two different kind of environments — there was the Air Force program and work going on at MITRE, and then what was going on in the academic world — or was there a lot of collaboration and interaction between those?

Denning: I'd say interaction. Roger Schell himself was from the Air Force and knew the Bell-LaPadula work.

Yost: He helped fund it.

Denning: Now I remember. Roger graduated from MIT in 1971. He helped Bell and LaPadula at MITRE under an Air Force contract shortly after that. The Bell-LaPadula model went beyond the simple types of access control that we'd all been looking at in Multics. The access matrix model talks about who is allowed to make a particular type of access to an object, for example, read or write. The Bell-LaPadula model defined access rights based on relationships between security tags of files and clearances of the user's program accessing the file. It addressed the concerns of government security with different classification levels and compartments. They showed how to tag objects with their security ratings, and programs with the security clearances of their owners, and decide how to limit access to avoid unwanted flows. They also showed how to compute the tag of a new object whose content is derived from other tagged objects.

The tagging system was an extension of the access control system. In the security world, a document is given tag at least as high as the highest tag of any paragraph in the document. Paragraphs are individually tagged. A paragraph cannot be added to the document if its tag is higher than the document's tag. A person can only read a document if his or her clearance is at least as high as the document's tag. Once the rules are known for deriving new tags, for reading tagged documents, and for modifying tagged documents, it is easy to translate this into an operating system structure for managing documents at different levels of classification.

This model defined allowable information flows between users and objects (such as files) in the system. Dorothy's work went farther. She actually traced flows. She was able to trace flows inside a program and write rules for the compiler so that the compiler put a proper security tag on data at any point in the program. It is possible that a secret program outputs unclassified information that can be proved to be derived only from unclassified inputs. The Bell-LaPadula model was not designed for that.

Yost: In 1976, you published "Fault tolerant operating systems."

Denning: Yes. I drafted that during summer 1975 when I was on a sabbatical visit to University of Cambridge, where I worked with Maurice Wilkes and Roger Needham. We had long talks on many issues about how to achieve fault tolerance and error confinement in operating systems. I met daily with Needham and we amassed a lot of ideas; Roger encouraged me to write them up. I did. I left Cambridge with a draft in hand, which eventually became that *Computing Surveys* paper.

My thinking in that paper was still strongly influenced by Multics, from which I inherited a deep concern for the structure of hardware and kernel architectures to support security, error tolerance, and error confinement. I think Roger Needham used some of the ideas from our discussions as part of the design of the CAP system that he was working on. I particularly remember a discussion we had about interrupt systems, and how to secure an interrupt system. Hackers were beginning to figure out how to attack a computer by tripping up the interrupt system. We figured out how to secure an interrupt system. And I think that went into the CAP.

Yost: Can you elaborate on that?

Denning: First, we had to get clear definitions of what we wanted from an interrupt system. From there it was easy to figure out the architecture to support the concept. The purpose of an interrupt system is to respond rapidly to "exception signals". An exception signal is a signal from a sensor indicating that an "exceptional condition" exists and needs rapid attention to be resolved. Exceptions include internal program errors such as divide-by-zero, overflow, underflow, array index out of bounds, and protection violation, and external signals such as disk completion, packet arrival, loss of external power, and alarm clock.

The idea is that the CPU should respond quickly to the exception signal. The CPU is wired up to check for an exception at the end of every instruction cycle. If the CPU detects that an exception signal has occurred, it is supposed to make an immediate procedure call on a special "handler" routine in the operating system that responds to that signal. The handler procedure activation record is pushed on to the stack of whatever program the CPU is running. When the handler is finished, it does an ordinary return, which restores control to the CPU within the interrupted program, exactly where it left off. This concept was called "unexpected procedure call" in operating systems of the 1960s.

There are nasty security vulnerabilities in this design. The most obvious is that the handler activation record goes on to the stack of another program, and if the handler itself has been hacked, it could access any other part of the interrupted program's stack or data memory. Another vulnerability is that the list of handler entry points – called the "interrupt vector" in operating system terminology – could be hacked, so that some of its pointers are to the hacker's programs rather than the intended handlers.

Roger and I concluded that we could secure this by using the protected entry point idea from capability machines. The interrupt vector would be a list of enter capabilities, and the machine would not let a hacker tamper with them. When an enter capability is used, the called procedure instantly gets its own capability list distinct from that of its caller, thus preventing a hacked handler from accessing the interrupted program's stack or address space. There were some details such as making sure the CALL instruction works properly with either an enter capability or an ordinary procedure entry point address.

Yost: What was the first capability machine on the commercial market?

Denning: I think it was the Plessey 250. Plessey Telecommunications is a British company. They were interested in the capability architecture because of its error confinement properties. They built for telecommunications switching. They claimed that this system worked exactly as they wished and was really good about detecting and confining errors. Their systems were very crash resistant. In earlier switching computers, crashes could leave errors in data that propagated around the system, doing damage until they were finally detected. By that point, it would take a lot of undoing to fix the damage and restore the system to good working order. Plessey said that the capability machine stopped the machine soon after the error occurred, making restoration fast and much simpler. They really liked their system.

Yost: Do you recall when that was?

Denning: Early 1970s. Plessey put their system into operation in 1969 and started talking about their positive experiences in 1970 or so. Plessey was not the only capability based system. There were several others in that era. Precursors to fully working systems were the Rice Computer (1959), the Burroughs B5000 machines (1961), the Dennis and Van Horn capability supervisor (1966), Fabry's Magnum machine (1967), and the Time Sharing system CAL-TSS at UC Berkeley (1968). After Plessey System 250, other working systems included the CAP computer at Cambridge (1975), the CMU Hydra system (1971), the IBM System/38 (1978), and the Intel iAPX 432 chip (1981). The CAP's operating system was too complex, IBM's system enjoyed a limited success and Intel's chip was a flop because it did not do ordinary things like procedure calls very efficiently. Still, there was quite a lot of interest. Henry Levy wrote a nice book about it called *Capability- and Object-Based System Concepts*.

Yost: In that 1976 article about fault tolerant systems, you wrote, "the apparent lack of vendor interest in capability machines results in part from the costly trauma experienced by the entry of the computer era into its third generation in the mid- to late 1960s." Can you elaborate on the costly trauma?

Denning: In the middle 1960s, several major vendors came out with what they called "third generation operating systems." These were systems with all the latest innovations including virtual memory, hierarchical file systems, multiprogramming, time sharing, interprocess communication, extensive controlled sharing, dynamic linking of separately compiled modules, and more. The US vendors included IBM, RCA (which since left the computer business), General Electric, Honeywell, Univac, and DEC. Trying to get all those new functions working together harmoniously in these new systems was really tough. Several of those companies dropped out of the operating systems business. Trying to persuade them to reengineer their systems around capability addressing was a monumental task. And remember that Cambridge CAP came to a pessimistic conclusion about the complexity of capability based operating systems. On top of that the chip world started moving aggressively toward RISC architectures around 1980, further dampening interest in anything that looked complex.

Yost: Can you comment on the CAL 6400 ?

Denning: No, not much, I was not close to that one. Butler Lampson at Berkeley played a major role in its design. He cut his teeth on that system. A lot of the protection ideas he brought to our OS core course task force in 1970 were hatched in his experience with that system. I recall that he experimented with a novel swapping mechanism, where code and data objects flowed through main memory like a FIFO pipeline and would be processed while they were in main memory. Objects that flowed out of memory would go into a disk queue to return to memory later.. This sort of automatic circulation of all the programs through the memory seemed to reduce the total amount of swapping and gave better performance. I know Lampson was an enormous fan of the Dennis and Van Horn paper and used some of their capability ideas in the Berkeley system.

Yost: So it's roughly the late 1970s time frame that you're really moving out of computer security research, is that correct?

Denning: Yes. I made my declaration with Dorothy probably around 1975, and said, "This is your area, you'll be the family expert on this topic. I'll be the family expert on operating systems, architecture, and performance evaluation."

Yost: Is it an area that, even though you're not conducting research, has been of continuing interest to you and that you follow?

Denning: Oh yes, it's been of interest. I have always been interested in the way the architecture could be organized to enhance security. With the RISC revolution, it seems that many people gave up the idea of hardware support to security and tried to provide security functions in software only. Many modern hacks, such as tricking the interrupt system into getting you into supervisor state, or buffer overflows, would be near impossible with just a little hardware support.

But my contributions to security literature dwindled away rapidly after 1975. Dorothy has done so well with that, beginning with a novel idea about lattices in her thesis, and then a long series of contributions in the data security, protection, and cryptography areas. Once she became intensely curious about what motivated the underground hackers. She tracked some of them down and talked with them to form her own conclusions. She was heavily criticized for “consorting with the hacker enemies.”

Yost: Donn Parker did some of that, too.

Denning: Yes. While Donn agreed with her desire to learn more about hackers and their motivations, he thought she was getting too cozy with them. She took that criticism to heart and soon went and spent a lot of time with the FBI and some law enforcement agencies, looking over their shoulders while they worked on information-system crimes, and helped them with her knowledge of the technology. I think Donn Parker was happy with this outcome. A lot of people admired her for being willing to go out and hang out with hackers to reach her own conclusions about them. I think she found many of them were misguided and had social issues, but she wanted to know that for herself.

Yost: Can you tell me how you became involved with CSNET?

Denning: CSNET? Oh yes. I began my involvement in the late 1970s. In 1979, I was appointed head of the CS Department at Purdue. One of the big issues on Purdue’s plate at that time was how to raise its stature in research. Purdue had a pretty good rating, but the faculty aspired higher. We had the impression that the front-line research of the day was being done at universities with ARPA contracts, which entitled them to a connection with the ARPANET. We thought that if we could join the ARPANET, we could get closer to that research, and perhaps get some of our own ARPA contracts. Purdue EE department had had an early node on the ARPANET, but had let it go. It seemed like a Catch-22 problem. We wanted the ARPANET connection back, but did not have any research proposals of interest to ARPA, and could not qualify because we were not connected to ARPANET.

In discussions with Larry Landweber, a Purdue alumnus and then chair of CS at Wisconsin, I learned that he and his faculty had similar concerns. Soon we learned that many of our peer universities in the Midwest had the same concern. We began to wonder what we could accomplish by banding together.

Landweber organized a meeting of interested parties at Wisconsin in 1979. He got Kent Curtis of NSF to come, and also Bob Kahn of ARPA. As agency representatives, they were interested in helping the rest of the CS university community get access to the networking technology. But Kahn said that under DoD rules, there's no way to do it without a research contract, and they had insufficient budgets even for those in the room. We wondered if there might be some way of leveraging a relationship between NSF and ARPA, so that NSF grantees could be granted access to ARPANET. Curtis was interested in this because he had been looking for ways NSF could help the CS research community, which had been bleeding from faculty "brain drain" as systems faculty left university for industry. That brought us to a discussion of how, with NSF support, we could create an ARPANET clone for computer science, with a bridge to ARPANET. Curtis and Kahn agreed this was a worthwhile possibility.

Landweber organized a group of us to draft a proposal to NSF for a CS network. That was a long process because Curtis wanted us to circulate the proposal widely in the community to ensure that there was broad community support for what we proposed. After two or three rounds, in 1981 we submitted to NSF a five-year proposal for a CSNET. It was subsequently approved by the National Science Board, which had to approve all large grants – CSNET was $5 million, large in those days. The PIs were Larry Landweber, Dave Farber, Tony Hearn, and me. NSF assigned Bill Kern as the full time program manager representing Kent Curtis.

The plan called for us to construct a network with different tiers of service from a dialin telnet service on the low end, to a phonenet email exchange service in the middle, and a full implementation of TCP/IP at the high end. We also needed to set up a consortium of universities to manage the network, set up working self governance, and be financially self sustaining by 1986.

NSF put us initially under the stewardship of NCAR (National Center for Atmospheric Research), the weather consortium. They wanted NCAR to teach us how to set up and operate as a consortium. They had long experience with NCAR and a lot of trust in them. This was marvelous help, because we had no idea how to set up and manage a consortium. NCAR completed its mentoring relationship with CSNET in 1983 or 1984.

In parallel with that, we had to design and implement a CSNET architecture suitable for the CS research community. Unlike the ARPA community, almost no one had an ARPA contract and any networking experience. The potential members of CSNET had wildly different idea of what they wanted from a network, from very low end and cheap connections, to very high end connections. Our CSNET had to work for all of them.

The mainstay of the initial CSNET architecture was Phonenet, a system that used the ARPA mail standard SMTP to exchange email between pairs of computers over a phone line. There only cost was the monthly rental of a data-enabled phone line and the appropriate connection fees. Dave Farber at University of Delaware had a mail client that was shared with everyone in CSNET.

For the high end, the only commercially available packet switched network provider in the US at the time was GTE Telenet, which used the European X.25 protocol. X.25 was based on establishing a connection with another server and passing streams of packets over the connection. X.25 was incompatible with TCP/IP, which was a connectionless datagram protocol. CSNET took on a project to build an interface that would accept TCP/IP traffic from ARPANET and route it to CSNET nodes through X.25. Doug Comer and I took that one on at Purdue.

At Wisconsin, Landweber and his team designed and built a name server, which was a directory service of all CSNET members. And at Rand, Tony Hearn set up the gateways that bridged between ARPANET and CSNET. Rand had an ARPANET connection that went in one side of the gateway, and the X.25 and phone connections went in the other side. The gateways flowed traffic smoothly and transparently between the two networks.

We also hired BBN to be the network manager. They helped new members install software, get connected, answered their questions, and collected their dues. In many ways, they were the Internet's first ISP.

Around 1983 when we were establishing the costs for the X.25 service, we discovered that the members preferred a fixed monthly rate to paying by the packet, as

GTE Telenet did. Even if the fixed monthly rate was higher than the average monthly packet chargers, it was easier for them to budget for. Years later the phone companies learned the same thing when setting cell phone rates: customers preferred a fixed monthly charge to a per-message-unit charge.

CSNET had to get some policy concessions from the ARPA and NSF sponsors. Kahn told us that DOD had a prohibition on any non-DOD entity sending traffic to the ARPANET. He had to figure out a way to get connected even though a gateway computer. He came up with the idea of writing an MOU with NSF for joint research on networking. Under the MOU, any university designated by NSF as a valid member of CSNET had permission to send traffic into the ARPANET. The MOU and that arrangement solved the inter-network connectivity issue. It was a major policy breakthrough. Eventually, they opened up the list of eligible CSNET members to include industry research labs such as HP Labs and IBM Labs. That was another policy breakthrough because it allowed network traffic from commercial entities on a government-sponsored network. By 1986, NSF was moving into a larger networking project, NSFNET, and these precedents with ARPA served them well.

By 1986, after five years, CSNET had achieved all its goals. In retrospect that was a rather amazing accomplishment. There were about 120 member university departments and research labs. The network served upwards of 50,000 researchers and students. It was completely self supporting, with each member paying an annual fee.

CSNET was a few years later absorbed into the NSFNET, which by that time was the backbone of the emerging Internet. Many CSNET alumni helped with the design and implementation of NSFNET. CSNET members welcomed this because they no longer had to pay dues to a CSNET consortium.

Yost: So the CSNET along with some of the regional networks contributed to NSFNET?

Denning: Right. CSNET contributed a lot of expertise and experience to NSFNET. The success of CSNET emboldened NSF to take on the much more ambitious NSFNET project.

Yost: Were there any discussions about network security with CSNET or did that come up?

Denning: That's always there; it's always been a concern. When you hook up into a network you no longer have any idea who's trying to access you. You need extra protections on your local operating system. CSNET certainly enabled and facilitated networking research. They worried about intrusions in networks, especially if the intruder is trying to hide by tracing through different nodes. They worried about malware, such as worm and viruses, and how to reverse engineer a captured malware. CSNET didn't sponsor research, CSNET was simply an organization that hosted a network. Some of its members proposed security projects to NSF and were funded through their universities. CSNET's existence supported their research and also set a context for their research to be meaningful.

Yost: Bernie Galler appointed you SIGOPS chair. Was there any formal subgroup within SIGOPS that was focused on computer security?

Denning: I got involved with the ACM special interest committee on time sharing, SICTIME, shortly after attending the first symposium on operating systems principles in 1967. They asked me to be newsletter editor, which I did for almost two years. They wanted to convert SICTIME to a SIG so that it could draw members from all over ACM and be a more formal organization. I helped them write the bylaws and I proposed that they expand from "time sharing" to "operating systems." Our proposal was approved by ACM Council in 1969 and Bernie Galler asked me shortly thereafter if I would chair the new SIGOPS. I accepted his invitation.

One of my first activities as SIGOPS chair was to organize the second SOSP, which we held at Princeton (my home university) in 1969. It generated a huge interest and we got an overflow crowd of attendees. That launched the SOSP series as a regular SIGOPS even every other year. I also cooperated with Peter Wegner, the chair of SIGPLAN, to establish conferences on the interface between programming languages and other subfields of computing. One of the products of that was the Principles of Languages and Systems conference. I don't know if it still exists, but I think ASPLOS, the conference on architectural support for programming languages and operating systems, is an outgrowth. Security issues were woven into all these things. There was no explicit subgroup of SIGOPS devoted to security. There was eventually a SIG on security that was put together in the early 1990s, but I have not tracked it.

Yost: When you were ACM president, I understand that NSA under Director Inman sought to restrict or review crypto publications. Can you discuss your perspective on this and what position did the ACM take in the near term and the longer term?

Denning: That was Admiral Bobby Ray Inman, who was director of the NSA. The NSA was very concerned about US-invented crypto technology getting out of the US and into the wrong hands that would use US technology to hide their communications. The huge interest in public key cryptography in 1975 stimulated a lot of research on security and use of cryptographic protocols in the emerging Internet. Up to that point, the NSA was pretty much the sole authority on advanced crypto systems. Public key cryptography started to change that.

Inman's initial position was that all crypto information was "born classified," analogous to atomic energy information. He wanted to make it extremely difficult for adversaries to benefit from advanced crypto systems developed in the US. He also wanted to set up a prepublication review process so that all draft manuscripts on crypto topics would be reviewed by NSA before being published.

Inman's publication review proposal created a huge uproar among researchers. They said they didn't want NSA telling them whether or not they could exercise their First Amendment rights. Inman saw that getting his proposal adopted would be extremely difficult, if not impossible, and this was going to be a big problem. And he established an entity called the Public Cryptography Study Group. ACM was one of the member organizations, and David Brandin represented ACM. A part of their job was to find a way to help NSA satisfy its concern about releasing sensitive crypto information without compromising research publication. They never found a common ground; the idea of any restriction on academic publication didn't go over with any academic and the idea of openness didn't go over with the NSA. The best they could agree on was a voluntary system. You, as an academic researcher, could submit a paper to NSA for a review at your choice. Some people chose to do this and many chose not to. But then Inman's tour of duty ended, and his public face disappeared. But his interest moved elsewhere and the whole thing died.

However, the tension between the government crypto community and university researchers has continued to this day. Now, a few university researchers have expressed that same old concern, because they see the dangers to critical infrastructure from unrestricted hackers. We diffused the argument back then, but didn't eliminate it; we never found a good solution to it.

Yost: And finally, are there any topics that I haven't brought up regarding computer security that you'd like to discuss?

Denning: I've been interested in what might be called a security sea change in the last five years. We see this here at NPS in the Navy. About fifteen years ago, Cynthia Irvine founded a computer security research center and associated curriculum, and has been its leader ever since. It mostly focused on information assurance, which was the government's major concern -- to make sure information is treated properly and not released into the wrong hands. Information assurance is a pretty broad issue. About five years ago, the DOD and Navy declared cyber security (their term for "computer, data, and communications security") a top priority and started talking about their worries over a "Cyber Pearl Harbor" if attackers exploited known vulnerabilities. The term "cyber" is now the shorthand way to refer to all these concerns and actions. A long time ago Donn Parker warned that professional criminals would join the hacker ranks; they are now here. They're not amateurs. In fact, many of our hackers are amateurs compared to the professionals out there.

A major concern now is developing our own professionals, our own people who understand the issues and can protect us. This is a massive manpower issue. The Defense Department is looking to hire thousands of security experts and universities aren't capable of producing them right now. They're trying to help, but there's a big gap. Five years go, security was a backwater concern, and now it is among the highest national security priorities. That's pretty amazing to me and certainly justifies our intuitions long ago that production and security of data were critical issues for an operating system. The problem was hard for people sharing a single operating system and has become a nightmare for people in the vastly interconnected network. The many doomsday scenarios we hear constantly remind us how fragile things are. So many things rely on the proper and safe functioning of the network. Our civilization depends on it.   If somebody could actually erase all that data, they could well throw us back into the Stone Age.

One of the doomsday scenarios is a bomb designed to deliver an EMP, that is, electromagnetic pulse. Such a bomb could fry any electronic circuit within its radius. It could disable all cars, computers, and all embedded computers inside a radius from wherever that bomb went off. With enough bombs in the right places, telephone switches could break down and Internet data centers would be disabled. Without communications and ability to recover data, businesses and government would not be able to operate, putting the whole foundation of our current civilization to the test. Locally we worry that in a massive power failure following an earthquake, the gangs of Salinas would fan out and pillage and loot Monterey and other wealthier cities of the region. The police wouldn't be able to stop them. Look what happened in New Orleans during Hurricane Katrina. When enough of the infrastructure is broken, the economy can collapse and it can take a really long time to restore it. True, this is just a scenario, and yet a lot of security people are concerned and do their best to have preventative plans in place.

Yost: Very sobering thought.

Denning: Let me tell a related story, which gives some hope that things would not get as bad as these scenarios picture. I go back to 1994 when I was editor of the ACM *Communications*. Two years before, ACM had declared a new *Communications*, a flagship magazine for all members rather than a research journal for a few. My editorial team and I were trying to learn how to be a magazine. One of our new features was interviews with interesting people.

With my senior writer, Karen Frankel, I interviewed Steve Jobs to ask him about Apple Computer and the development and evolution of the technology. Karen and I had a prepared set of questions. When we finished the list, there was still time. I asked Steve what he thought of our fears of hackers and intruders taking down the Internet, like the Morris worm had nearly done in 1988. I asked, "What do you think, is this a real possibility?" Jobs went silent and he buried his face in his hands. He stayed in that pose, unmoving, for at least three minutes. For an interview, three minutes is an interminable silence. Karen and I started whispering, "Has something happened to Steve? Should we call for help? Maybe he's had a seizure and needs a doctor?" He was just sitting there transfixed in some state and was not communicating. I think I might have touched his arm, but he did not respond. Just as we were on the verge of running out of the conference room to call for help, he suddenly came out of it. He looked up, and said: "No."

His answer to the question, "Do you think this is a serious threat?" was a simple, authoritative no. Karen asked, "Why do you say that?" He said, "Because they need it to do their work. The last thing they're going to do is shut off the means of doing their work." That was the end of the interview.

Recently, when Job passed away, I reflected on this incident and saw something I did not see before. I remembered that Jobs had a reputation for using a skill of meditation that he learned in India for doing his own mental focus group and figuring out how users are going to react to technologies. He doesn't need to hire focus groups, he does it in his own mind. I think what we witnessed Steve Jobs doing that. He spent three minutes in an altered state, imaging himself being hackers doing their everyday work. He saw that they could not get their work down if the Internet were shut down. That would make no sense to a hacker. So what I think we witnessed Steve Jobs doing a meditative mental focus group in his mind. And it was borne out that since the Morris worm, and since that interview, there has never been an incident where hackers shut down any significant portion of the Internet.

Yost: That's a fascinating story.

Denning: Back when we talked to Steve Jobs, the Web was not fully developed and there were very few data centers. Today there are hundreds of data centers around the world. Last year, The Economist estimated that three per cent of the world's electricity is used to power data centers. That does not include any of the other costs of running the Internet such as the routers and links. That's massive. Almost all business data is in at least one data center. No wonder security people worry about attacks on data centers. Might it be possible to take out a few of them and inflict massive damage to business and commerce?

This loose collection of data centers is called the "cloud." When you put data into the cloud, you are putting it on a disk in some data center. When you get service from the cloud, you are talking with a server in a data center. A lot of engineering has gone into those centers to reduce vulnerabilities. Data are replicated and stored in many centers in different geographic locations. The centers have independent power generation and are heavily shielded and fortified. Their locations are undisclosed. A lot of work has gone into protecting the data centers, they are critical infrastructure. Google has become so good at engineering data centers that they are able to take full snapshots of the entire Internet and process them into gigantic indexes that enable any Google search query to be answered in under half a second. No one knows how vulnerable the data centers really are. There has never been a major data center outage. That engineering is paying off.

Yost: Is there enough redundancy to counter the threat?

Denning: The engineers think so. A data center contains thousands of processors and disks – sometimes as simple as a warehouse full of PCs on a local network. They replicate data within the same center and with more copies at other centers. If disks fail in a center, the data can be recovered from the same center. If a center fails, the data can be recovered from another center. The stakes are big. The data centers are well engineered and well guarded.

Those little beginnings we started 60 years ago in operating systems and databases have blossomed into the backbone of the world's commerce. Pretty amazing.

Yost: Without question. Well thank you so much. It's been extremely helpful. Our interview is done.

Denning: It was my pleasure.