An Interview with

BUTLER LAMPSON

OH 452

Conducted by Jeffrey R. Yost

on

11 December 2014

Computer Security History Project

Cambridge, Massachusetts

Charles Babbage Institute
Center for the History of Information Technology
University of Minnesota, Minneapolis

Butler Lampson Interview

11 December 2014

Oral History 452


## Abstract

Turing Award winning computer scientist Butler Lampson briefly discusses his education and work in time-sharing with Project Genie, the Cal Time-Sharing System (Cal Computer Center), and the Berkeley Computer Corporation (BCC), as well as his seminal work at Xerox PARC (systems and graphics work to create the office of the future—the ALTO and the basis for the Xerox STAR—the primary achievements that led to his receiving the Turing Award in 1992). The interview, part of an NSF-funded CBI effort to document computer security history, concentrates on Lampson's many contributions to the computer security research field, and his broader perspectives on various aspects of computer security developments (including the economics of computer security). Lampson talks about his work to build a capability machine, and expresses that despite considerable interest from some research scientists, this is not a fruitful path for computer security. He explores the context to his important note on the confinement problem. He also discusses the context of his access matrix. Finally, he discusses his work at Digital Equipment Corporation (DEC) and Microsoft, including his work at DEC on distributed system security, and Microsoft's Palladium Assurance stack.



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, from the Charles Babbage Institute at the University of Minnesota, and I'm here today on December 11, 2014, at Microsoft's New England Research and Development Center in Cambridge, Massachusetts with Butler Lampson. This is an interview for CBI's project funded by the National Science Foundation, "Building an Infrastructure for Computer Security History." Alan Kay conducted a very useful oral history with you for the Computer History Museum, which really spanned the many areas of computing where you've made major contributions. With this interview I'd like to focus heavily on computer security, but I will begin with a few biographical questions. Can you tell me when and where you were born?

Lampson: I was born in Washington, D.C. on December 1943.

Yost: I understand you attended the Lawrenceville School, outside of Princeton.

Lampson: That's right.

Yost: And had an opportunity while in high school to use an IBM 650?

Lampson: That's right.

Yost: Can you talk about that experience?

Lampson: A friend of mine, Dave Loveman, a classmate, had found the 650 at Princeton, which is just six miles away. It was not being very heavily used and he cut a deal with the woman who controlled it that if it wasn't being used for any other purpose he could come up there and play with it. I found out about this and thought that it'd be cool, so we both played with it for I think about most of a year. We didn't do anything particularly significant but that was my first actual exposure to computing. There was no security.

Yost: And you went to Harvard and studied physics.

Lampson: Yes.

Yost: Can you tell me about that decision and your evolving interests at that time?

Lampson: I had always been very interested in physics and mathematics. So I was, at that time when I was an undergraduate, I was pretty committed to becoming a physicist. I took a lot of physics and math courses and did a moderate amount of computing on the side while I was an undergraduate. There weren't many courses, but I did take a few courses and I also worked for a physics professor, writing programs to interpret his spark chamber photographs.

Yost: And that was at the Cambridge Electron Accelerator…

Lampson: That was at the Cambridge Electron Accelerator Center, right; which blew up just a few months after I left Harvard. [Laughs.]

Yost: I've read about that.

Lampson: Yes, it was designed to handle an explosion; they were using bubble chambers at the time, bubble chambers full of liquid hydrogen, so it's actually quite dangerous if it gets loose. They had carefully designed the building so that if such an accident did occur, the roof would rise up and relieve the excess pressure. I believe it all worked like a charm [laughs] but I'm not sure they ever got the CEA working again. A few years after that, the NSF or the DOE, or whoever was paying for it, decided that they had too many little accelerators and most of them were shut down.

Yost: That facility was a joint M.I.T./Harvard project?

Lampson: I believe so; I was not paying attention to such administrative details at the time but that seems plausible.

Yost: Did you make any connections to M.I.T. and M.I.T. faculty during that year or so?

Lampson: I don't think I had any connections with the faculty. I used to come down there quite regularly to use the reading room because Harvard's computing was much less strong than M.I.T.'s. In fact, I still remember very well the old Aiken Computation Laboratory, which was a relatively new building. It had a big lobby, maybe 20x40 feet, and you walked in the door and you were in this lobby. It was fairly dimly lit. Down at one end there was a receptionist desk,

which was lit up, and all along the right hand side of the lobby there was a huge glass wall. On the other side of that wall was the machine room, which was also quite dimly lit. And in the machine room, all down one side — the machine room was, I don't know, 40 by 100 feet, a big room. All down one side was the Mark I Relay Calculator — racks full of relays and cables that looked like they were a foot thick, and typewriters and all kinds of paper tape readers for reading the program paper tapes that were two-and-a-half feet wide. All kinds of things. All turned off, of course. And all down the right-hand side was the same thing again, that was the Mark IV Relay Calculator. Also turned off. Way down at the end of the room — you could just barely see it — there was a pool of light, and in that pool of light was the university central computing facility, which was a UNIVAC I. Pretty obsolete machine by 1960. And then, I think, in the fall of 1962, they came in, swept all this stuff away and installed an IBM 7090, and then it was no longer romantic. But Harvard was one of the institutions—I think it's pretty much true that all of the U.S. universities that were deeply involved in the first wave of computer construction, they all fell back into the second or third tier, with the exception of M.I.T. Harvard was one of them, with Aiken's Mark I and Mark IV. But the same thing was true at Illinois, and UCLA, and Princeton. M.I.T. was the only one that continued to be in the front rank. So Harvard was pretty much moribund in computing by the time I was there.

Yost: Did you have any awareness of the CTSS?

Lampson: Oh, yes, because I did come down to M.I.T. quite frequently, and CTSS and LISP were the two hot things at M.I.T.

Yost: You decided to pursue a Ph.D. in physics at Berkeley?

Lampson: Right.

Yost: What year did you finish at Harvard?

Lampson: 1964.

Yost: And you started that year at Berkeley?

Lampson: I spent a summer at Los Alamos as an intern, the summer of 1964. I was supposed to be doing physics-related things, but I got captured by a group that wanted a programmer. [Laughs.] So I spent the summer futzing around with a giant FORTRAN program. But yes, in the fall I was in Berkeley. I hooked up with three other physics and chemistry graduate students. We had a four-person apartment close to the campus and two months later, two of my four roommates were arrested in the first free speech demonstration, so that was exciting. [Laughs.]

Yost: Exciting time to be in Berkeley.

Lampson: Exciting to be in Berkeley, just so.

Yost: I know you recounted this story in your other oral history but it's a great story, so can you tell about how you first learned about the Genie Project from Steve Russell?

Lampson: Sure. The 1964 Fall Joint Computer Conference, which was the main computing conference at the time, happened to be in San Francisco so it was easy for me to go. I went and I ran across Steve Russell, whom I knew very casually, and he asked me how's Peter Deutsch doing? And I said who's Peter Deutsch? He explained to me that Peter was an undergraduate at Berkeley who'd done a lot of work with computing at M.I.T., where his father was a physics professor. Peter started in, I believe, at the age of about 11. And when he went to Berkeley he'd gotten connected to this Genie Project, which was an attempt to build a medium scale time-sharing system on hardware that was more than an order of magnitude cheaper than the hardware on which CTSS had been done at M.I.T. The project was housed behind an unmarked door in the southeast corner of Cory Hall, which was the electrical engineering building. So he told me how to find it, and I went over there, and went through the unmarked door, and there was sort of an air lock, and then there was another unmarked door. Went through that, and there was a moderate sized room with a few desks in it. No people. Another door on the far side of the room. I went through that door and there was an extremely large room. Cory Hall, at the time, had very high ceilings because when they built it they didn't have enough money to build all the space they wanted but what they did have enough money to do was build very high ceilings with the intention of putting in mezzanines later, and that had not happened yet. It was done some time in the 1970s, I think. And in the middle of this very large — and this room had various odd bits of computing equipment scattered around it — in the middle of it there was this rather small-looking computer and a young guy sitting at the console of it reading a paper tape. After he finished reading the whole paper tape, he wound it up and read it in again. I said to him, what on earth are you doing? He said it's a two-pass relocatable loader. And I said what?! And he said

yes, yes, I know; I'm rewriting it. And that was my introduction to Peter Deutsch. [Laughs.] And we worked together on the Genie time-sharing project, and later at BCC, and later at PARC; altogether for I guess about 15 years.

Yost: And at that time, what faculty were involved with the project, overseeing it?

Lampson: Well, there wasn't a lot. Nominally the project was headed by Dave Evans, but he wasn't really paying terribly much attention to it, and not too long after that, he went off to Utah to start his big graphics project there and to found Evans and Sutherland Computer Corporation. There was Harry Huskey, who had built one of the first computers that was built at a university, in the early 1950s, I think. But he was pretty much retired at the time [and] was not paying much attention. He was actually my thesis advisor, but he didn't give me any trouble. [Laughs.] And they had hired an assistant professor named Wayne Lichtenberger, basically to have more faculty involvement in the Genie Project, but the Genie Project was really run by a graduate student named Mel Pirtle. Aside from that, at that time, there really wasn't a lot of faculty involvement in computing at Berkeley. I think there was a math professor named René De Vogelaere, who was interested in numerical analysis, and I had some contact with him.

Yost: Can you talk a bit about your transition from physics to computing; you were obviously captivated by this project.

Lampson: Right. Well I'd always been sort of interested in computing and spent quite a bit of extracurricular time on it. But this was the first time I'd really had a chance to do anything

significant, and I was totally captivated by it. In addition to that, I was starting to feel somewhat uncertain about my career as a physicist because my theory about theoretical physics—I'm no good as an experimentalist. I can still remember, I took a course at Harvard that was suppose to introduce physics students to electronics, and I still remember that I was working in the lab on that course one Saturday afternoon and picked up a soldering iron by the wrong end. That made it pretty clear I was not cut out to be an experimental physicist. And theoretical physics, I felt at the time, and I still feel — although I don't know as much about it now as I did then — that in a sense, theoretical physics is a fairly narrow subject at any given time. There are relatively few major things to work on, so if you don't think you have at least a shot at a Nobel Prize, you probably aren't going to make any significant contribution to the field, and I didn't think I was that good. So that was a push away from physics, but the attraction to computing was much stronger.

Yost: Can you tell me about the earliest discussions you remember among the group regarding computer security on time-sharing systems?

Lampson: In those days, we were working on time-sharing, as were people at M.I.T. and Carnegie Mellon, and a few other places. And we didn't so much think of it as security, but the whole idea of time-sharing was that you had a bunch of independent people who didn't necessarily share the same interests, who were all sharing the same piece of hardware, and it was considered to be quite important, partly for the kind of reasons that we think of today when we talk about security but mainly just for pragmatic reasons, to keep them out of each other's hair. You didn't want the other guy's bugs fouling up your program, and so everyone who designed

time-sharing systems paid a lot of attention to trying to design them in such a way that when you were running a user's programs, there was no way that those programs could interfere with the operation of the rest of the system, outside of the little virtual world that the system had created for running that user's programs. And it was just taken for granted by everyone that that was important. As I say, to some extent for reasons that we now think of as security, but mostly just for pragmatic reasons.

I remember hearing a story when I was at the M.I.T. Computation Center. They had a consulting service, which consisted of two ladies in a room across the hall from the computer center, and one of these ladies was pretty nice and pretty knowledgeable, and the other one was not very nice at all but much more knowledgeable. So if you had to go to the consultants, you thought about whether your problem was easy or hard, and if it was easy you went to Marianne; and if it was hard, you went to Judy and took the necessary pounding that you would probably get. One day, a guy walks into this room, and he doesn't understand this at all so he picks Judy randomly. He walks up to her desk and slaps down his listing, and says, I want my money back. She says oh, why? And he says, I ran this program yesterday and it ran perfectly; and I ran it today and it didn't work, so there must have been something wrong with the computer. And she says oh? You ran exactly the same program today that you ran yesterday? That's right, exactly the same. Why did you run it again? Of course, it had different input today. [laughter]

But it was considered to be very high priority that the behavior of the machine should be predictable. And as I say, what we now think of as security was essential for that, in that world where everyone was sharing the same hardware. So people just took it for granted, and almost all

of the early work on security was focused on trying to make it easier to do sharing of both the hardware resources and information that was being stored in the computer system without compromising this property.

Yost: Can you talk a bit about the division of labor between you, Peter, and Melvin Pirtle on working on the system?

Lampson: Mel was basically the hardware guy. He was responsible, a) for running the project, and b), for figuring out what we were going to do in the way of hardware. The basic idea of this project had been to make relatively modest changes to a stock computer, a Scientific Data Systems 930, to make it possible to run time-sharing on it. So there had to be some sort of memory protection scheme, and there had to be some provision for making sure that user programs couldn't execute instructions that could mess up the system. So that was Mel's role. And Peter and I sort of traded things off in writing the system software for the thing. We both would work on different parts of it at different times. And I think Peter had to have had a lot more experience in this area than I had, because he had worked on the PDP-1 at M.I.T., which was another hands-on machine. And the owner of that machine was Jack Dennis, who was a young professor at M.I.T. at the time, who introduced the basic ideas of security, based on capabilities, into the field.

Yost: Had you studied CTSS much at all and did that provide any influence on the design of the system on the Genie project?

Lampson: We had all read the manual, and so we had a pretty good grasp of what you could do in CTSS. I don't know that; it's hard for me to say. Many of the lower level details were very different in our system. In some respects the hardware that we had to work with was quite a bit more flexible than the CTSS hardware. And certainly we had much better control over it, because the SDS 930 was a relatively small machine and it was built by a company that sold their equipment to engineers, unlike IBM, which basically sold their equipment to either big computer centers or businesses. And very often what the customers for the SDS machines would do would be to hook the machine up in some more or less complicated way to some piece of external equipment, non-standard external equipment. So SDS had made a lot of provision for messing around with the machine. They didn't really anticipate that people would go in and hack the memory system, but they did have all the necessary components and documentation and so forth to make that easy. So in that respect, it was very different.

Yost: Was SDS a company that provided ongoing support, once their machines were installed?

Lampson: You could buy a maintenance contract, but I don't think we had one, and they probably would've been very reluctant to maintain it after we'd gone into the guts of it and torn it apart, but we had plenty of expertise to maintain it ourselves. And as I said, there was excellent documentation.

Yost: You did much of the documentation for the system, can you discuss that effort?

Lampson: We needed manuals, and I wrote what seemed to be unavoidable, let's put it that way. I think Mel was pretty strong on manuals, so he applied pressure when that was necessary to get something written. I think all that stuff is currently online; you've probably seen some of it.

Yost: Was there an expectation that this system might be used in a number of other places, at that time, once complete?

Lampson: I am a little hazy on whether that was part of the original plan, but that became a goal pretty soon, and that was for a variety of reasons. Of course, we thought that what we were doing was great and we wanted to world to benefit, but also, ARPA was very interested. This project was funded by ARPA, and ARPA was very interested in getting time-sharing out there, which meant that it had to be something that you could buy, because very few people were in a position to build and maintain their own. So ARPA very strongly encouraged our efforts to persuade somebody to actually make this a product. It was fairly difficult, because the management of SDS didn't believe that there would be much of a market for it, and so they dragged their feet for quite a long time before they finally agreed to do this, but in the end I think they made a lot of money on it.

Yost: You completed your dissertation entitled, "Scheduling and Protection in an Interactive Multiprocess System."

Lampson: That what it was called? Oh, okay. [Laughs.]

Yost: And it had meaningful computer security components —

Lampson: I can't remember the security part of it at all. I remember the scheduling part of it pretty well.

Yost: Well it was one of the very first dissertations that had any discussion at all related to computer security, and really predates others in that regard by a half decade or a bit more; Dorothy Denning's Lattice Model dissertation in the mid-1970s was entirely focused on computer security, and others later that decade.

Lampson: Yes, that could be, I guess there must have been several of them at M.I.T. in the early to mid-1970s. Trying to think of where else there might've been dissertation work; maybe Schell.

Yost: Roger Schell's really wasn't computer security. He had already developed an interest in the topic, but his dissertation was on dynamic reconfiguration.

Lampson: Well M.I.T. had a sizable project in the early to mid-1970s to try to figure out how to make Multics more secure. And, of course, various aspects of security had been fairly important in the original design of Multics, for the same time-sharing reasons that I discussed earlier.

Yost: Right. I'm very aware of the faculty members, Corbató and Saltzer, who were publishing in this area, but I'm just not aware of any doctoral theses.

Lampson: Right. Dave Redell wrote a thesis on revoking capabilities at M.I.T. in 1974, I think. I'm just trying to think; not sure when Jerry Popek's VM thesis was written. That had a fair amount to do with security, although it wasn't the primary focus. Yes, we'd started to get more interested in that, beyond just the most basic question, how do you keep user's programs out of each other's hair parts of it.

Yost: Basing the system on capabilities —

Lampson: That's another thing I wanted to say; most of the work I did on security while I was at Berkeley was not done on the Genie Project, was not done on the 940, it was done on the Cal TSS system, which is a capability-based operating system that we built as part of the computer center, supported by the computer center. The computer center had acquired—with NSF money, I think—a couple of CDC 6400s, and I think part of the deal with NSF was that one of these machines would be used for production and the other would be used at least part of the time for one form or another of computer systems research. And what we undertook in the project that was led by myself and Howard Sturgis, and involved several other people who subsequently became fairly well known, we undertook to build a complete capability-based time-sharing system. And of course, security was a major focus of that system. To my knowledge, this was the first capability-based system that actually had users, and we learned a lot from it although in the end it turned out to be a bust. In fact, I think it was the first of a number of capability systems, all of which have been busts.

Yost: Can you talk about any prior research that influenced your thinking about capability systems?

Lampson: My memory of it is that we thought it up from scratch. I think Bob Fabry by that time had written about his ideas about capability systems, too. But I don't remember seeing that stuff at the time. We started this in 1965 or 1966, I think, and it ran until 1969 or 1970, something like that. And Jim Gray wrote a paper about it; he was one of the people that worked on it part time, and Charles Simonyi worked on it. Simonyi did Bravo, and went on to Microsoft and made billions. Bruce Lindsay subsequently became pretty well known for working on transaction systems. Those were all people that worked on the Cal system. Howard and I wrote a paper on it, but that was a number of years later as a sort of a spinoff of Howard's Ph.D. thesis, which is a description of that system.

Yost: That's the 1976 CACM paper?

Lampson: That sounds plausible. I think Howard's thesis was published a couple of years earlier. But that was long after work on the system had ceased.

Yost: At the time, what did you see as the key attributes of a capability system for computer security?

Lampson: I think our basic idea was again, not so much the kinds of things that people tend to think about today when they think about security, which is how do we keep the bad guys out, but

more using capabilities as a tool for making the whole system more reliable, by bounding the bad effects of bugs. So our model wasn't so much there's some evil group in Russia that's trying to break into the system, as there are two user programs running and we want to make very sure that even though they're communicating in fairly complicated ways, one of them can't have any effects on the other except the ones that are intended by the receiving party. I think that was very much the focus of all the early work on capabilities, up to and including, for example, the Cambridge CAP System, which was done in the mid-1970s, which was also a bust.

Yost: Wasn't there a project that Fabry consulted on at the University of Chicago on the Chicago Magic Number Machine?

Lampson: I believe that's right. I knew nothing about that project at the time and I'm very hazy on how far they actually got. You probably know a lot more about that than I do.

Yost: The machine was never built.

Lampson: Okay. Well one of the things about our project, of course, was we didn't have any hardware support for capabilities, it was purely done in software, unlike the Magic proposal and the CAP machine at Cambridge, which had a lot of hardware support for capabilities. But I remember when the CAP guys were more or less done, and they actually had their system running and they had users. I said to them well, I believe the main experimental hypothesis for this system was that with capabilities the system would be a lot more robust and there'd be a lot fewer bugs. How'd it work out? Their first answer was, we have no idea, which seemed quite

horrifying. It's clear they'd gotten swept away by all the intricacies of system construction and just totally lost track of the original hypothesis. But then they went back and they looked at all their logs and they said well, we found about half a dozen cases in which there were bugs which we were able to find quite easily, and we think they would have been very difficult to find if we hadn't had the capability support. As a result of that, they and I both concluded that the thing had been a bust, because the effort required to do the capabilities, and the cost that were incurred in performance and resources expended, were totally disproportionate to making it easier to find half a dozen bugs. And I believe that's been pretty much the lesson of every attempt to introduce hardware capabilities. I know some attempts are continuing today; perhaps the DARPA CRASH program; or have until quite recently; I've sort of lost track of its current status.

Yost: Peter Neumann's DARPA funded effort [pause]

Lampson: Yes, Peter never gives up. He clings to the faith, absolutely. But he's never built a system that's actually worked. We can talk more about this later; it probably would be more appropriate. But this is one of these things that people get religious attachments to, and I believe it's in complete defiance of the evidence. In spite of this, of course, capabilities have been very successful in operating systems, it's just that they're not called that, they're called file descriptors. UNIX has them, and every major operating system has them. The difference is that they're not the foundation of the security system, they're basically a performance optimization; that's slightly oversimplified, but once you open a file, you get a file descriptor for which the security is controlled some other way. You get this thing called a file descriptor, and that then

gives you permission to do things to the file, and as long as it stays open, you can pass it off to other processes, and so on and so forth. That's exactly a capability, but it's not long-lived.

Yost: M.I.T.'s PDP-1 time-sharing system that Jack Dennis worked on, was that influential at all?

Lampson: It was, in the sense that Peter [Deutsch] worked on it. And so, for example, the forking primitives that we had in the 940, I believe, were very heavily influenced by that system, and maybe other things as well. I didn't actually know that much about that system; there wasn't terribly much documentation for it. But as I say, Peter had worked on it so he knew a lot about it. So yes, I think it had a big influence.

Yost: As you're doing this work in the late 1960s, there's the first recognition of, the earliest articulations of, the multilevel security problem. Willis Ware and Bernard Peters did a paper in 1967 for the Spring Joint Computer Conference.

Lampson: Right.

Yost: That leads to Ware's leadership of the committee and the Ware Report in 1970. Were you aware of this in the late 1960s?

Lampson: No, I wasn't aware of any of this at all. I first ran across this in the early 1970s and I was kind of struck by it, and it led me to write my paper on the confinement problem. But I

believe I went to some meeting—I think you mentioned 1971—where I was first exposed to this, and it seemed quite interesting intellectually. That's another thing that's been a bust, is attempts to actually do that, as I'm sure you've heard a great deal about that. [Laughs.]

Yost: I definitely have.

Lampson: Yes, it was another one of those things that seemed like such a good idea — capability seemed like a great idea, too — it's just that they don't pay their way. And multilevel security is worse because it tends to—maybe it would pay its way if it would actually work, but what happens in real life is that everything just goes to the highest possible classification level.

Yost: I came across something that indicated that Needham and Wilkes visited the Chicago team on the magic numbers.

Lampson: Oh is that right? I never heard about that.

Yost: Did they visit Cal as well, before doing their CAP system?

Lampson: I certainly knew Roger by that time, and I'm sure we talked about these things because I think I knew about the CAP project at the earliest time. I don't know. I do remember that David Wheeler, who was in the computer lab in Cambridge and a close colleague of Wilkes, was on sabbatical at Berkeley in 1966, 1967, something like that. He actually engineered the piece of hardware that we used to connect the teletype terminals to the CDC 6400 time-sharing

system. So he would've known all about the work that we were doing. Although that's not his primary interest he certainly would've known all about it. I cannot remember whether either Wilkes or Needham actually visited Berkeley; and I can't remember when I first met Roger, either. Probably was at some SOSP.

Yost: As I understand it, one prototype was built while you were at Berkeley.

Lampson: Prototype of?

Yost: Of the SDC 940 time-sharing system, the BCC 500.

Lampson: In the late 1960s I was involved in three different systems: there was what eventually became the SDS 940, that was the Genie Project system; there was the software that we built to run on the CDC 6400 with extended core storage that was in the computer center at Berkeley, and there was no hardware development there except for this terminal connector gadget, which was not, I mean we had to have that because we had to have terminals but it was not central to the project at all; and then there was BCC. Those three things were completely separate from each other. They shared a lot of people, and of course the reason we started BCC was because we wanted to build a second generation system as a follow-on to the 940 at the university, but we discovered that it was basically impossible to do that because you could not marshal, within the constraints of the university bureaucracy and process and so forth, you couldn't marshal the hardware resources necessary to do that. And so we went around looking for computer companies to partner with and we found one, Scientific Control, but it didn't work out. The

whole thing didn't really make commercial sense, as we learned when we did the startup. But then we couldn't help but raise venture capital and start this company, in which we got to try out all the great ideas that we'd been incubating in the aftermath of the 940 system. It was the classic example of when you do a startup, the most important property a startup has to have is there should be no technology risk, because there are so many other risks. There are so many ways things can go wrong, if the technology can also go wrong it's just too much. We had a lot of technology risk and we went through $4 million of venture capital money, which is quite a lot for 1970, and eventually we built a working machine. But from a commercial point of view, it came to nothing.

Yost: With BCC what was the source of the venture capital?

Lampson: There were two $2 million chunks, as I recall. The first chunk came from a computer leasing company called Data Processing Financial and General, which was thriving in the late 1960s, and they were looking around for opportunities to expand. Mel talked them into the proposition that if they would sponsor this company, then there would be these great time-sharing systems and that would be the wave of the future. And they could expand beyond their other boring business, arbitraging IBM's sales and rentals prices, which is what the computer leasing business was all about. And then when that money was used up, Mel managed to talk the venture capital fund the University of California had started into putting up another $2 million. There was no venture capital industry at the time.

Yost: Did SDS provide any support?

Lampson: No. Even though they made a lot of money on the 940, they were very negative about time-sharing systems in general, and they had their own axe to grind. No, our next contacts with SDS came as a result of the fact that Xerox had bought them.

Yost: SDS had sold a time-sharing system to Tymshare, by that time?

Lampson: That was the 940; that was the system that we built at Berkeley. It was basically our system with some fairly minor modifications. They sold, I think, 40 or 50 of them. They sold a bunch to Tymshare, they sold a bunch to Comshare, and they sold individual ones to quite a few research establishments.

Yost: So that sounds like they were making a good deal of money off the 940.

Lampson: Yes, they made quite a lot of money on the 940, but they never really believed in it; it was just too weird for them. And when they built their next generation family, the so-called Sigma family, they did put in essential things that you needed to have if you wanted to do time-sharing, but they never took the software aspects of it terribly seriously. I think that was mainly because there wasn't really that much demand for it from their hard core science and engineering customers. So, I think if they had seized that opportunity and done everything right they could've made a big success out of being a time-sharing company, but that wasn't in their DNA, and it definitely wasn't what they did. We'd have been happy to collaborate with them rather than

doing this startup, but they were not interested at all. It's probably just as well, too, because it probably would've bombed because our aspiration level was way too high.

Yost: In another interview you stated that nobody you worked with on the Cal time-sharing system wanted to build another capability-based system. You touched a bit on this, but can you elaborate on why you and others involved felt this way?

Lampson: I think our conclusion was that the gain just wasn't worth the cost. There's a fundamental structural problem with capabilities, which is that — I'm sure you've seen my access matrix picture of subjects and objects, as Peter Denning rechristened them — and so if you think about how you're going to implement that, you can organize things by subject. You can make a list of all the objects that a subject is allowed access to; those are called capabilities. Or you can organize it by object; you can make a list in one form or another of all the subjects that are allowed access to this object; that's so-called access control lists. For long term security, the second one is the only one that's going to work. Why? Because the question you always want the answer to when you're looking at the system is who can get to this object? And it's very difficult to get that answer out of a capability-based system, but that's almost always what you care about. You know, here's the file full of Sony salaries and I want to know who can get to it. And so file descriptors work okay, because they're very transient. The file descriptor goes away, certainly when the system is rebooted and usually much, much sooner than that, so it doesn't matter terribly much that you can't keep track of the fact that once the file's open and you have the file descriptor, there's no way to tell which of those are floating around. If you really wanted to know that, because all the file descriptors are kept in memory, you could arrange to find that

easily enough. But for the long term storage this stuff has to be kept on the disk, which means that you can't search through all of it in any reasonable way. And as I say, the basic question you want to answer for anything that goes on for more than a few hours, is who are the people that can get to this resource? So it has to be organized that way; there's just no way around it. At least this is my view.

And then the question is, are capabilities a good tool for organizing the system? And I think our conclusion there was they work, we made it work, and they have some advantages because you get uniformity. By contrast, the way in which security works in UNIX, for example, they have these file descriptors, those are capabilities, but they also have a bunch of more *ad hoc* things. And on the whole, I think that works better. Some of them are more complicated, but you can tune some of the *ad hoc* things that need to have high performance better, specific requirements. I don't know really how you would figure this out. You'd have to do some systematic experiments, and they would have to on a fairly large scale; and no one has ever attempted that kind of thing. Basically, I think our judgment was that there were easier ways to get to the same effect. The experimental hypothesis—it is true that the system that we built was extremely reliable, and as I said when I was talking about the CAP system, many, maybe most of the people who worked on capability systems thought of that as the most important property. But there are other ways to build systems that are plenty reliable enough; maybe the capability-based ones were a little bit more reliable, but it didn't really matter that much. So add all this up and the answer is it just doesn't work out. It's not that it can't be done; you can even build a perfectly serviceable system this way, but the payoff isn't big enough to motivate you to do it.

Yost: Do you know whether SDS was focusing much at all on security in marketing the 940?

Lampson: No. They were focused on time-sharing, and as I said before, security is an essential part of time-sharing, at least to the point that you want to make sure that one user's program can't inadvertently mess with another program. And of course, when the 940s were actually sold, they sold into commercial service bureaus. It was actually pretty important for the security to be better than that because there was no strong control of the users. As far as I know, if you went to Tymshare and said I want time-sharing service, they didn't make any attempt to guarantee that you wouldn't be on the same machine as a competitor. So in that sense, they were starting to move in the direction of the kind of security that people worry about today, but that was not very central; people didn't worry about that too much, I don't think. And of course the systems were much, much simpler than today's systems, so there were a lot fewer opportunities to screw up. The main concern, as I keep saying, of these capability-based systems was not so much to make it secure in the modern sense, but to make the consequences of bugs less severe.

Yost: Drawing on JOSS, you wrote a programming language called CAL. Can you talk a little bit about that?

Lampson: It had nothing to do with security. I had an idea, which was that you could do a JOSS-like system where the user could change individual lines and run the program any time, and yet you could compile the program down to actual machine instructions, which was not what JOSS did—JOSS ran an interpreter over the program. In CAL the program would not be as efficient as one written in FORTRAN, at least the way I implemented it. Each statement had to be translated

completely independently of the others, so you couldn't do any cross-program optimization, but it would be much, much faster than an interpreter and I thought this was a cool idea. So I built CAL, partly because it seemed that it would be nice to have a system like that, and partly to see if the system would actually work, which it did.

Yost: And along with Peter Deutsch, you wrote QSPL?

Lampson: That was the system programming language, right. That was sort of our version of C, where the focus was a little bit different. C grew out of BCPL and BCPL's main interest was control structures. QSPL grew out of a system that Ken Knowlton did at Bell Labs called L6, where the main interest was in data structures and how you could efficiently pack things together; pack the components of records together into machine words, so we had much better facilities for that than C did, and the control structures were not as good. But from 10,000 feet they were the same thing.

Yost: The BCC500 pioneered shadow page and redo logs, can you discuss that?

Lampson: Shadow pages and redo logs were on CAL TSS. We didn't do those on the BCC system.

Yost: Right.

Lampson: Those were Howard Sturgis' invention. That was the scheme for making the file system crash tolerant, and it was subsequently copied in various forms in lots and lots of database systems, in some cases with direct influence, because Bruce Lindsay and Jim Gray, who both worked on CAL TSS, went off to IBM and worked on the System R Project, which was the first relational database system. But in the end those techniques got superseded by so-called redo logging that Mohan invented. Shadow pages became discredited, but that was many years later. Howard invented those things in the quest for a strategy to make the file system as robust as the rest of the system was. In the file system, the main thing you worry about is the possibility that a crash might corrupt it.

Yost: To return to BCC. In 1971, the company is essentially shut down?

Lampson: Right. It survived a little bit longer than I stayed there, because Bob Taylor came along and swept up the core of engineering talent in late 1970, and we all started at Xerox PARC in January of 1971; at least that's what I remember. The company survived a little bit longer. It survived long enough for ARPA to cut this deal with Mel Pirtle, by which he would go off and try to save the ILLIAC IV Project, and they would give the University of Hawaii a big enough contract to do research in time-sharing that they could buy the BCC machine for enough money to pay off the creditors. At least I'm pretty sure that that's what happened. Nobody ever actually told me that, but I can't think of any other explanation for what happened.

Yost: Can you tell me about the transition for you to joining PARC?

Lampson: Well, we got to put all the big system complexities of BCC behind us, and there we were without any real significant commercial constraints. Xerox had given our part of PARC a very open-ended charter, which was to invent the office of the future. They didn't know what that meant and neither did we, but we had a lot of ideas about interactive computing and we proceeded to put them into practice. I wrote all about this in the paper on the software of the Alto.

Yost: Can you talk about the specific roles that you and others played on the Alto project?

Lampson: I was responsible for the system software. Chuck Thacker and Ed McCreight were responsible for the hardware; I only kibitzed on the hardware. I did make one minor contribution, which was to figure out how to reduce the number of micro cycles required to fetch a word of bitmap and give it to the display, from seven to six, which sounds trivial, but it meant that if you were filling a whole screen with display, with this change the machine would run about 30 or 40 percent faster, because most of the machine was consumed feeding the display at that point. So it made a big difference if it was only six cycles instead of seven because it meant that instead of using 75 percent of the machine for feeding the display you were only using 65 percent. But other than that, I wasn't much involved with the hardware of the Alto. I was very much involved with the whole sort of strategy, and I did write the original version of the operating system. And then Charles Simonyi and I developed Bravo, which was probably the most successful single application — most successful and most influential single application.

Yost: And that was the first what-you-see-is-what-you-get text editor?

Lampson: That's right.

Yost: And you also were involved with Local Area Networking research and development?

Lampson: Only peripherally. Charles Simonyi and I started this project that built what was essentially a local area network version of the ARPANET, as it then existed. Without thinking about it too carefully, we decided okay, the ARPANET runs off 50 kb phone lines, we'll build a local area network that will be much like the ARPANET but it will run over 50 megabit lines, which was unthinkable for a wide area network in those days. But, of course, that would've been wild overkill for the actual need, and there would've been many complexities. We did quite a bit of design on that, but we never got it as far as actually implementing anything. And fortunately Bob Metcalfe came along with his ideas for Ethernet, and that was cool. I guess I kibitzed some on that; I kibitzed enough that I actually got my name on the patent, but I can't actually remember what I did. But there was no doubt in our minds that we had to have a local area network, because after all, the slogan was "personal distributed computing," and without networking you can't have distributed computing. And also, we had an extremely clear priority and that was to print. We only had one laser printer, so that meant that you had to have a network, otherwise you wouldn't be able to get the printing to the printer.

Yost: So with the idea of distributed computing was computer security considered?

Lampson: No, we didn't think about that at all, quite deliberately. We said that's not important. We've got other things—we've got bigger fish to fry. So the Alto had no support for supervisor mode, or memory protection, or anything like that, and we didn't worry about that in the basic Alto software at all. We did when we had a central file system, we did the standard kind of login protection for that, but that was absolutely it.

Yost: So you and a small team are doing this pioneering work, creating the office of the future. How did you think of yourself? Did you still think of yourself, in part, as a specialist in computer security in those years?

Lampson: I never thought of myself as a specialist in computer security. In fact, I don't believe in computer security as a specialization. My view of it is that it's a branch of computer systems, and the people who specialize in it are not as good as the people who do it as a sideline. I think you could just look at the history of the field over the last 40 years and you could pretty much see that. I continued to have some interest in it. I wrote that paper on the confinement problem, and Howard and I wrote a paper about the CAL system. So I continued to have some interest, but it wasn't a primary focus.

Yost: Can you provide me with some more context of the *Communications of the ACM* 1973 short paper on the confinement problem?

Lampson: I wrote that paper because I had gone to some meeting where I had heard about multilevel security and the difficulties that were caused by Trojan horses. If your threat model is

that bad software can get its hands on your data, which is not something that anyone had ever thought about before these DoD-influenced guys started to get into it. Certainly the people that worked on time-sharing systems never thought about that. As I said, the motivation there was to keep programs out of each others' hair, not keep data in. It seemed very interesting, so I was turning it over and I was motivated to abstract a little bit from what we had talked about at this conference, whatever it was.

Yost: Had you read the two volumes of the Anderson Report?

Lampson: I can't remember. My guess is no; my guess is that I was just told about it, but I have no clear memory of it. I never had it on my shelf until I found it on the net a few years ago.

Yost: And then the second paper you mentioned in those years, "Reflections on an Operating System —

Lampson: That was about CAL TSS.

Yost: With Sturgis.

Lampson: Right.

Yost: Can you tell me about the context for doing that paper and why you chose to take a retrospective look, and analyze it?

Lampson: We'd done all this work, and aside from Howard's thesis it was never documented, and it seemed like we'd actually learned a lot about quite a few things, some of them having directly to do with capabilities and some of them having to do with other things. It seemed extremely worthwhile to make all that hard-gained knowledge public, and I didn't have the usual "you've gotta publish so you get tenure" motivation. Berkeley had already given me tenure before I'd left. But it just seemed silly to let all that die, and it didn't seem likely that that many people would read Howard's thesis, and in any case I wanted to say things in a different way than in Howard's thesis. I didn't have much control over the thesis, but I wrote most of the paper.

Yost: Can you elaborate on some of those different ideas?

Lampson: Well, besides the capabilities, I think it's fair to say, one of the interesting ideas in CAL TSS was the shadow pages and redo logs stuff for the file system. And the other thing was we had a very heavy emphasis on extensibility.

Running out of something?

Yost: Just checking battery level.

Lampson: So we were very proud of what we called the F-RETURN mechanism, which was a scheme by which you could take a low-level API in the system and turn it into a higher level one

by essentially packaging it up with an error handler that would catch errors that the low-level guy detected, and send control off to some higher level piece of code without incurring any performance cost in cases where the low-level thing succeeded. And we used that a great deal in the system to — you can sort of think of it as a super-generalized form of page faulting. There's no generality in the page faulting mechanism, but the whole idea is you fetch a word from memory and if it's there, it goes through at full speed. And if it's not there then you get a trap and you can get control and potentially do something potentially much, much more complicated in order to make it work. We were trying to generalize that to the point where it worked for anything. Similarly, we had extensibility mechanisms — the details of which I can no longer remember — that worked at higher levels in the system. This was all part of our general notion that we wanted the system to be highly modular so that it could be more robust. We wanted it to be highly modular in a safe way so that we could make a reliable system out of a bunch of components that could be designed and tested individually. I think it's fair to say that was really the driving force behind the whole design, manifested in the capabilities, and the redo logs, and the F-RETURN mechanism, and a bunch of other things. They were all in the service of reliable modularity.

Yost: What was the general response to the publication of that paper?

Lampson: I think a bunch of people read it, but for the most part, people thought oh yes, here's another failed operating system. [Laughs.] As far as I could tell, it never stopped anyone from trying to build another capability-based system in the future.

Yost: And what about the response to the note that you wrote on the confinement problem?

Lampson: Well I think that that's had much more influence actually, although it has far less content. I mean it's not really anything — there's a bunch of new ideas in the CAL paper, there's not really any new ideas here except I think I articulated the distinction between what I called storage channels and timing channels in a way that no one had done before. I'm not absolutely certain about that, but that's what I think. I personally wasn't that interested in this. I wrote that paper because it seemed like a mildly interesting intellectual problem and I wanted to sort of tease it out and make it publicly accessible in a way that wouldn't require you to read all the detail that went into the Anderson Report, or for that matter, to look a little farther into the future, when Bell and LaPadula did their work. That was much less abstract. My goal was to try to tease out the essentials of non-interference, as it's now called, in a way that was independent of implementation mechanisms. So I think that paper's had a lot of influence. Although that whole subject became somewhat moribund until [Barbara] Liskov and [Andrew] Myers sort of reinvented it 10 or 12 years ago.

Yost: Do you know if it had any early initial impact on the trajectory or the way things were going with the Air Force research?

Lampson: I do not know, because I was not really involved. Aside from this one meeting I went to, I was not really involved in any of that stuff, at least not that I can remember. I suppose I must have had some connection, because some time in the relatively early 1970s, ARPA asked me to be on a review committee for Multics to try to figure out where they should go with it.

And they were certainly thinking very seriously about the security aspects of it at that time, so I must have had some contact with it at that time, but I don't remember anything about it. And as I say, I thought of all this information flow control stuff as being intellectually interesting, but I was never terribly convinced that it was practical. Although I have to say that I did not fully anticipate that the Orange Book initiative would crash and burn as decisively as it did, because I did not understand the commercial — not quite the right word but I'll use it — commercial aspects of it. The way I think about it is that DoD said, we're only going to buy Orange Book systems after 1984 or whatever it was. And when 1984 came, a purchasing agent had two choices. They could play by Orange Book rules, or they could get a waiver; there were many, many advantages to getting a waiver, and that's what they all did. [Laughs.] By the time I joined DEC in 1984, Steve Lipner's A-level VMS Project was just coming to fruition. And it was relatively soon after I joined DEC that the company decided not to ship it because their marketing people said, quite correctly, that you wouldn't sell enough to pay for all the support cost that would be incurred, for precisely this reason.

Yost: Right.

Lampson: By that time, it had become quite clear to them, and I guess Steve was convinced because he didn't really push back terribly hard on that decision; some of his people did.

Yost: If I remember my interview with Steve correctly, he indicated that he agreed with the decision and that he in fact was central to making the decision that supporting it would cost more than the revenue it could bring in to DEC.

Lampson: I believe that's correct. At the time that project was started, it wasn't so clear that everyone was going to follow this waiver strategy, but by the time it came time to actually decide to ship it, I think it had become fairly clear. But I know there were people that were extremely bitter about that, and feel that great opportunities were lost, and all that sort of thing, which I'm quite sure is not true. It was a very nice piece of engineering, but there was no market for it. And there still isn't any market for security, by the way. [Laughs.]

Yost: Major corporations accept vulnerabilities exist. . .

Lampson: They made that abundantly clear, right. My current slogan is "retroactive security," which means that instead of trying to stop bad things from happening — not that you should completely abandon stopping bad things from happening — but instead of depending on stopping bad things from happening, you should make sure that when bad things do happen you're in a position to mitigate it either by punishing the bad guy or by undoing the undesired state changes. My view is that this is the way that security works in real life. You know the reason that your house doesn't get burgled is not that the burglar can't get through the lock on the front door — the lock serves a purpose, it keeps out the riffraff and it makes it clear that the guy is doing something that he's not supposed to — but any competent, any even incompetent burglar can get through the lock without any trouble, right? The real reason that your house doesn't get burgled is that the risk of getting caught is too great. It may be small, but it's too great, and when that stops being true, burglary goes way up.

Yost: And meaningful punishment is, of course, important?

Lampson: And there has to be some sort of punishment. Of course punishment can take many forms. There can be fines, there can be social ostracism, there can be getting fired from a company, lots of different ways in which you can punish lots of different kinds of misbehavior. So it seems to me that between that kind of deterrence, based on punishment and undo, that's the way security works in real life. You know, most people think the security of the financial system, for example, depends on encryption and vaults and all that kind of thing, which is completely untrue. The security of the financial system depends on the fact that almost any transaction can be undone. Sometimes the undo is fairly expensive and painful, might even involve a trip through bankruptcy court, but in cases where it can't be undone you have to be much, much more careful.

Yost: In 1975 you were a principal designer of the programming language Euclid, primarily funded by DARPA, to write systems programs that could be verified. Can you tell me about the motivation for doing that project and what was achieved?

Lampson: Ever since the days of the CAL system, I'd been sort of abstractly very interested in the question of how you could make systems in which maybe you could have some confidence. Program verification was in its very early stages, and it seemed pretty clear that it was going to be very hard to verify programs written in assembly code, or C, or BCPL, or QSPL, or whatever it was that you had at that level. And in addition to that, we had been working on a language code called Mesa, where the primary focus definitely was not on security, it was on modularity,

but there were some connections. This seemed like an interesting thing to work on. I'm not quite sure what got us started, actually. Probably it was people getting together at some conference, but I can't remember how the Euclid project got started.

Yost: What did you see as the most important elements of Euclid that allowed for programs to be verified?

Lampson: There were two major things. One was hardnosed type safety, which typically people didn't do in programming languages because it was too expensive. The other was the notion that you were completely explicit about which parts of the state could be changed by any procedure. So we had this import mechanism that allowed you to explicitly declare which parts of the data a procedure could see and/or modify. And that was in response to the fact that at that time — and this is still true to a considerable extent —one of the big problems with verification of an imperative language is figuring out how much damage a particular program, a particular procedure could do it if it's called. And so in support of that the other thing we did was we put into Euclid a mechanism called "collections" that essentially allows you to bound the scope of pointers so that you don't just have an undifferentiated heap, any part of which could be accessed by any pointer, but instead you can have sets of pointers and you can explicitly say, this is the collection of pointers that this particular procedure might mess with. And then the language is guaranteeing that it won't mess with anything else. The idea is that this would make verification significantly easier.

Yost: I found a reference to Euclid being used at MITRE and SRI.

Lampson: Could be. I was not paying attention. I'd be a little surprised, because the only implementation of Euclid that was done, was done in Toronto. As with all such systems you pay a price for the security, and not too many people are motivated to pay that price. If you want to be really hard-nosed about it, the price is not gigantic, but it's not small.

Yost: In both some articles and talks you commented in general terms on what could be understood about the economics of computer security. Did that become a focus of any researchers to do real analysis?

Lampson: Not in those days. Nobody paid any attention to those things, as far as I know. I think the whole idea to think about the economics of security is only about 10 years old. If there were people thinking about it longer ago than that I don't know about it.

Yost: You were part of a small team that designed Modula-2+, an extension of Niklaus Wirth's Modula-2.

Lampson: Yes.

Yost: Can you talk about the design of Modula-2+ and the elements that advanced security of Modula-2?

Lampson: Modula-2+ was basically a follow-on to Mesa— Modula, in fact, was Niklaus' version of Mesa, where he took the ideas from Mesa and stripped them down, because Niklaus was on sabbatical at PARC twice and that was where he got the ideas for Modula and also for the Oberon workstation project. He worked pretty closely with Chuck Thacker and some other people; Ed McCreight and some other people who were at PARC. So when we left PARC and went to DEC we were able to cast off all the legacy trappings of Mesa and start over. We said we don't want to spend five years designing and implementing a programming language. What can we do; Modula was a very comfortable place for us to start because first of all, it's not very complex, and secondly, it has all this heritage that we were familiar with. So we asked ourselves what are the things we could add; what do we see as the current needs for a serious programming language?

Yost: As I understand, by the early 1990's, this led to a team of Roy Levin, Chris Hanna, and Yuan Yu developing the Vesta system-building system.

Lampson: That was quite different, that had nothing to do with Modula.

Yost: Nothing to do with Modula-2+?

Lampson: Right. That was a completely separate enterprise to try to figure out how to do the assembly of large software systems in a reliable way, and systematically manage all the different versions that grow up, even in the kind of research environment that we were used to where

people were constantly changing bits of the system. But that didn't have any code—it was programmed in Modula but it didn't have any code in common.

Yost: In 1983, the incredible team that came to PARC and did so many amazing things to advance personal and interactive computing, breaks up and a number of you at the senior level go to DEC's Systems Research Center.

Lampson: Yes.

Yost: Can you tell me about that transition, and compare and contrast the research environments of PARC and DEC's Systems Research Center?

Lampson: Yes. That happened because Xerox decided to fire Bob Taylor, which was probably not a very good idea, and a lot of people got very angry as a result of that. There was already some outflow from PARC into the startup scene at the time. So for example, John Warnock and Chuck Geschke had already left to found Adobe, but this caused a mass exodus, and most of the people went with Bob to the new research lab. It was, in some respects—well, DEC was a computer company. Xerox is not a computer company. Xerox had given us this wide-ranging charter to do the office of the future. On the other hand, by 1984 it was pretty clear that the initial attempt to do that had not been a commercial success. I don't know if you've read any of the books about that, *Fumbling the Future*, the *Dealers of Lightning*, right.

Yost: I have.

Lampson: In my view, they don't actually get the story quite right. Certainly, *Fumbling the Future* doesn't. The story's a little ironic because Xerox said okay, we're gonna take this PARC stuff and make it into a product, and started an organization called the System Development Division, SDD for short, run by Dave Liddle to build the necessary hardware and software to make into a product. A few people from PARC went into that organization to get it started. And the people that went, for the most part, had a grand vision of the system they wanted to build. They wanted something that was fully integrated and where the user interface was quite uniform across a wide range of applications. In general, the whole thing was going to fulfill a wonderful dream of what office automation software should be like. And somewhat to my surprise, they actually succeeded in building the system. It took them four or five years, but in 1981 they shipped the Star system. There was only one thing wrong with it: it cost too much. I actually predicted about a year before they shipped it that it wouldn't work, and I was completely wrong. It did not fail because it didn't work, it failed to a very small extent because it was too slow, but it failed to a very large extent because it was too expensive.

The ironic part is that some of the people left behind at PARC, who didn't join SDD, were trying to convince them to build something much less grandiose that could be sold for much less money. But they would not have been able to fulfill their grand vision, so they brushed off all such suggestions. So it was sort of the opposite of the usual story you hear about how the researchers are in their ivory tower and the product people have to bring them down to earth. Here, in my view, it was exactly the other way around. The entertaining thing about that whole story is that Apple made exactly the same mistake a year and a half, or two, later with the Lisa,

which was also a commercial failure and also for exactly the same reason: it was too expensive. It had the same sort of integrated goals and it had a hard disk, and everything you needed to carry out that vision of what we'd been used to at PARC, but it was not something that you could actually afford in the technology in the early 1980s. It was only when they did the Macintosh, where they scaled back the aspiration level substantially, and even then the first successful Mac was the 512k Mac, which had, including its ROM, about five times as much memory as the Alto, and a smaller screen. That was the first time you were able to actually make this work.

It was part of the religion of the Alto that there would not be any integration. Each Alto application had to stand on its own feet, and there was a very simple reason for that. There was a lot of argument about that, but there was a very simple reason for that, which was that any interesting app used up the whole machine and there were no resources left for integration or commonality, or anything like that. The operating system was carefully designed so that you could throw away any parts of it that you thought you didn't need and either roll your own or do without. And all the major applications did that, because otherwise they wouldn't have fit. The only exception to this was Smalltalk, where they decided to pay the price of a full-blown object oriented virtual memory system, so they didn't have the physical memory constraint and then they were able to structurally do much more interesting things. But on the other hand, they could only build toys because it was too slow.

Yost: Were there any discussions on the trajectory of the cost of memory and Moore's Law, that even though costs were above what the market could bear that would improve?

Lampson: Some, we were certainly very sensitive. I mean that was the key thing that made the Alto happen in the first place, was when Chuck realized that memory was now cheap enough, at least in a research environment, that you could actually afford to give up one bit of it for every pixel on the screen, which had never been true before. We certainly were always very clear on the fact that the stuff we were building would be commercially viable 10 years in the future, but when it came to the Star system they were about five years off.

Yost: So when you arrived at DEC, within a different part of the corporation, Steve Lipner and Paul Karger's project VAX/SVS was underway. Did you have any involvement with that?

Lampson: Not really. I chatted with them a little bit, but not really. They were basically done by that time, I think; certainly all the design was set. I have a vague memory that I had some involvement with the DESNC project, I don't know if you heard about that.

Yost: No.

Lampson: This was a project to — what's the right way to say this? Think of a multilevel secure system where each user process is a separate physical machine and they're all hooked up by Ethernet. If you wanted to be multilevel secure you have to have some way of controlling the way in which all of these guys use the Ethernet, because otherwise anybody could talk to anyone else and you wouldn't have any control. So what the DESNC did was, you plugged your workstation's Ethernet into the DESNC, and it encrypted everything. The keys for the encryption were controlled in a centralized way so that your machine could only talk to other machines that

were approved by the DESNC central control mechanism. So they used cryptography to take this basically insecure communication system, the Ethernet, and turn into something where you not only could keep the communication secure but you could also control who could talk to whom. This was that product. I don't know how many of them they sold.

Yost: I was not aware of that.

Lampson: I was certainly; I don't know whether I actually had any direct influence of the DESNC, but I certainly was aware of it. This was sort of an extension of something that Stockton Gaines had proposed in 1973 or 1974, I think, where the question was, you want to do sort of coarse grained time-sharing on a supercomputer. But you want to have the same security guarantees that you have on a real time-sharing system, even though you can only run one job at a time on a supercomputer because the jobs are big and they consume all the resources. And you don't want to rely on a lot of complicated hardware and software to keep the jobs out of each other's hair. What you do is, when you want to switch from one job to another you take the whole state of the previous job, you encrypt it, and you dump the encrypted state on a disk somewhere, and then you take the encrypted state of the new job, decrypt it and load that into the machine. So again, the idea was to use encryption to keep the jobs out of each other's hair in such a way that you didn't have to trust very much hardware and software, assuming that you had confidence in the cryptography. I don't know whether Stockton's idea was ever implemented, but it was written up in something or other, sometime in the mid-1970s.

Yost: So around the time you joined DEC, the first version of the Orange Book comes out. You talked a bit about that, but do you recall what your initial reaction was to the criteria and how these standards defined and written up?

Lampson: I have a vague memory that I was casually paying attention to it, but not to the point where, for example, I ever sat down and actually read every page of the Orange Book document. I've never had more than an intellectual interest in all this multilevel stuff, and as time goes on I tend to believe in most of it less and less. You know, I actually think information flow control is a wonderful idea, but I'm not in favor of high assurance. For example, Microsoft Office has this thing that they called "information rights management." I don't know if you know about this, but I can lock up a document with IRM and I can say who's allowed to access it. The way that that's implemented is that the document gets encrypted, and then I can take the encrypted document, stick it on the file server, somebody else can grab and download it. And if you then try to open it up in Word, Word will talk to some server that will try to authenticate you as the user that's running Word, and if the server's satisfied with the authentication it'll give back the necessary decryption keys. This actually is used quite widely within Microsoft, and I think for a lot of corporate customers, it makes people feel much better about the security of Office documents. For example, if Sony had been doing this it would've been a lot harder for the North Koreans to do their thing. But the problem with it is that in many cases it gets in the way of getting work done, because something will go wrong with the decryption scheme and I can't get to the document. Or maybe the guy that wrote it made the wrong decision about who should be allowed to open it. Lots of decisions in which it can screw up. My view has always been that it would be a lot better system if the default were that if the IRM says you can't get to this document, there

were an override button that I could push that would say dammit, I want to get there anyway. Then you as the author would get a message saying that that happened, but there would be no enforcement of the restrictions. That would be much better for most purposes. Maybe not for everything, but for most purposes it would be much better.

In general, that's the way I feel about all that. This is completely inconsistent with NSA's goals but if you think about what information control might actually be useful for, the obvious use of it in an enterprise environment is when I send stuff outside the company, did I really mean to do that. [Laughs.] Today, Outlook actually puts up a rather inconspicuous note that says this address is outside of your organization, but it could be quite a bit more aggressive and protect the thing with cryptography. But again, you'd want a way to override it under almost all circumstances.

Yost: Outside of high assurance in the commercial realm, what was getting adopted essentially by the late 1970s, a number of corporations adopt systems like RACF and ACF2, and Top Secret; some systems go through certification but those that did, I don't think any of them qualified for higher than C2, at least not in their early years.

Lampson: I wouldn't expect so, because those were built by IBM for use by IBM's enterprise customers and IBM didn't give a shit about what the NSA wanted. [Laughs.] It was plenty good enough for the enterprise customers.

Yost: I learned that a number of corporations purchased these packages because the Foreign Corrupt Practices Act required them to have a system.

Lampson: That must've been much later.

Yost: That was in 1977.

Lampson: When was the Foreign Corrupt Practices Act passed? It was in the 1990s, wasn't it?

Yost: No, this was 1977.

Lampson: Oh wow, I didn't realize it went back that far. Okay, interesting. Often these things are done for theater or compliance reasons, not for actual security.

Yost: I talked to a number of people who were with corporations that purchased these commercial access control systems but . . . [pause]

Lampson: Just didn't actually use them. Right?

Yost: Right.

Lampson: But also, I think that in many cases those systems were fairly well matched to the needs of a big company to make sure that only the financial people could get to the financial records and so on and so forth. It's pretty coarse grained; big groups of people given authority; big chunks of data. The result of that is that the whole state of the security system is sufficiently

small that you can have a centralized management organization that can look at that and actually figure out what's going on. It's actually the opposite of the fine grain protection that everyone says they want. The fine grain protection doesn't work because you can't administer it. So even if by some wild chance you get it right, it's going to be wrong next week. So there's all this talk, people invent all these scenarios where it's crystal clear that people need to able to control access to each cell of a spreadsheet, but you can't implement it. I mean, you can implement it technically but you can't administer it. I wasn't very impressed with RACF when I first heard about it, but after a while I think I figured out that it was actually very well matched to the actual needs of the customer.

Yost: SHARE was all about, or rather IBM managers attending SHARE, were all about getting feedback and listening to customers.

Lampson: Right, absolutely, and IBM's always been very good at that. They're not very good at listening to their university customers, but that's because the university customers aren't a big source of revenue. [Laughs.] On a distantly related subject, have you run across the book, *The Limits of Strategy*?

Yost: No, I haven't.

Lampson: It's by a guy named von Simson. This is a guy who ran a consulting company, which basically acted as consultants to the CIOs of the Fortune 50. And for 30 years, he went around the computing industry several times a year, trying to find out everything that's going on, and

then tried to boil it down and communicate it to these CIOs in meetings that occurred a few times a year. So a few years ago he wrote a book telling the story of how this went between roughly the mid-1960s and the mid-1990s. It has lots of scuttlebutt about all the computer companies, you know, four or five chapters on IBM as it evolved, several chapters on DEC. [It] touches on almost everything else that happened that would conceivably be of interest to any corporate CIO, and has lots of very interesting insights into things from a very different perspective than what you get by talking to me, or Alan Kay, or Steve Lipner. It doesn't have a lot to say about security, I don't think, but it's a pretty interesting book and it has a lot of history, as I say, from a very different slant.

Yost: Great. I'll have a look. I am doing another a project on the history of computer services, so it will be of great interest. So I take it you weren't attending specialist meetings in security very often, meetings like the IEEE Security and Privacy Symposium in Oakland.

Lampson: No, I only go to those meetings when they ask me to give a talk. You know, as I said before, I basically don't believe in security as a separate topic of research and development; I regard it as part of systems. But we did do a lot of work at DEC SRC in the mid-1980s on distributed authentication. And we invented, I think in many cases, the first versions of stuff that has since become sort of generally accepted practice.

Yost: Did you work with Ron Rivest on some of that?

Lampson: Later. The work t we did with Roger Needham, Andrew Birrell, Mike Schroeder and I in the mid-1980s, shortly after we joined DEC, evolved into what we called the Digital Distributed System Security Architecture, which was a joint effort between some of us in research and two or three of the DEC product groups, the networking and the operating systems groups. We published a paper about that in, I don't know, 1988 or 1989. Something like that. And then we continued to elaborate all that stuff at SRC up to 1992 or 1993, when we built the system called Taos and worked out the whole story of what it means to communicate trust from one part of a distributed system to another. People have been reinventing that stuff for that last 25 years. [Laughs.] Sometimes they actually read our paper; sometimes they just reinvented from scratch.

Yost: I think there's a fair amount of that in computer security generally

Lampson: Reinvention. It's true in computing in general.

Yost: Yes.

Lampson: In this case, it seems to me somewhat less excusable, because after all, we did publish all this stuff in fairly prominent places.

Yost: Right. People should do literature searches.

Lampson: People should've been able to find it. That's right. And it wasn't as though it was all theory, either. We actually built a system and deployed it.

Yost: In the mid- to late 1980s, John McLean gave a paper at the IEEE symposium on computer security — on System Z, a critique of the foundations of Bell-LaPadula — created debate in the field and ultimately helped to lead to a conference on foundations for computer security that McLean was a part of for many years.

Lampson: And still going on, right? My colleague Martin Abadi goes to it regularly. I've only been to it once, when they asked me to give a keynote.

Yost: Were these developments things that you paid close attention to?

Lampson: No, not much. I've always taken the view that security is…sometimes there are things that are intellectually interesting but for the most part — what's the right way to say it? The basic problem with security is the customers don't care about it, which means most of the things you can think about it are actually never going to happen because there's no market for them. Maybe the customers *should* care, but they don't. And the reason for that is that people only care about bad things that have already happened. And even then, they're very good at saying well, that happened to the other guy because he was sloppy, but I'm not sloppy and it's not going to happen to me. And, you know, the other thing is in cases where people are doing more careful analysis; I'm pretty confident it's a rational business decision to not have security. Look at credit card verification. The way we do it in the U.S.

Yost: It's much less secure than elsewhere in the world.

Lampson: You know why that is? Why did we not deploy chip and pin when the Europeans were deploying it? It's full of bugs, by the way. But leave that aside—you can read a bunch of papers from the University of Cambridge that explain all of the bugs. Well, not all of the bugs, but some of the bugs. And it has a lot of bugs for the same reason that all these international standard mumbles have a lot of bugs, it's just that they're too complicated and there's too many different ways to configure them. But you understand why we don't have it? It has to do with the cost of phone calls. In the U.S., for a business to place a local phone call is much cheaper than it was in Europe. I don't know if it's true today, but it was definitely true 10 or 15 years ago. So it made economic sense in the U.S. to deploy VeriFone terminals and do online verification of credit cards. In Europe that was too expensive, so they never had that infrastructure. Then when chip and pin came along, there was a reasonable way to use it in offline mode. Nowadays I think people mostly do it in online mode, but being able to run it in offline mode was a critical part of the whole thing. The economics were different and in many cases, the same banks and certainly the same Visa and MasterCard organizations took different decisions in these two places. And I'm pretty confident it was for rational economic reasons, because you know they have a lot of information about how much credit card fraud there is and how much it's costing them and yaddayaddaya. And they bear almost all of the financial burden, although maybe the consumer is suffering inconvenience. But you don't suffer financially if your credit card is misused.

So that's a good example of how they definitely decided, even after this technology was fully developed and had been deployed in Europe for a long time, we're not gonna do this in the U.S., it doesn't make economic sense. Nowadays, when Target can be ripped off for 70 million credit cards that then have to be changed, that changes the economic calculation. You could say, well that was totally predictable, but no way are people going to think that way, because so many things are predictable and only a small fraction of them are actually going to come to pass, right? [Laughs.] These are the iron facts of security in the real world, but my guess would be that very few of the people you've been talking to have much interest in these facts. [Laughs heartily.] Certainly, very little of the academic work on security pays any attention to any of this.

Yost: Another area of computer security that is identified largely in the report written by James Anderson in 1980 that circulated, and by the mid-1980s, there's pioneering projects started on intrusion detection; SRI's IDES Project evolves into NIDES; a number of projects at National Labs, as well as some startup companies getting in this area.

Lampson: Intrusion detection.

Yost: Yes. Is that an area that you've paid much attention to? And those are systems that at least in the defense and intelligence communities, systems were developed and were deployed.

Lampson: Yes. I've never done any work in that area. I've looked at such things occasionally, and the general impression I get is — what was it? I think it was Jeff Jonas who I was listening to, he was one of the briefers for the National Academy's panel on signals intelligence that I was

on. And he said there are two kinds of people who work on standard security defense. There are the people who basically only aspire to catch the dumb guys, and there are the people who aspire to catch the smart guys as well. And it seems to me that intrusion detection stuff is only for catching dumb guys. It's not that difficult to evade it. I believe the only reason it's gotten so much attention is there's a huge amount of pressure to do something, and intrusion detection has the same property that firewalls have, which is that it does not require you to mess with the life of individual users. But if you want the security to actually work, there's no way of avoiding messing with the life of the individual users. However, for the most part, it's a ground rule that you can't do that. So, no, I haven't paid a lot of attention to this except every now and again I dip into it a little bit. But it just seems hopeless to me. The number of different kinds of legitimate behavior that you observe is so great that only the bozo is going to get caught by one of these intrusion detection systems. It's the same thing as malware detection by signatures. It's been known how to write polymorphic code for 30 or 40 years and guys that are good are going to do that, and you're not going to find them just with signatures. There's just no hope. You're only going to find the bozos. And nowadays, because of the internet — I don't know this for sure but I would imagine —there are toolkits out there that do polymorphism so you don't have to understand anything about it, you can just buy the toolkit and your thing will be magically polymorphic.

Yost: Some highly skilled malicious computer hackers, including some that have served prison terms, later go on and become consultants or employees within industry.

Lampson: Yes.

Yost: In talking to a number of people in computer security, I've heard divergent opinions on whether that should occur or not.

Lampson: You mean whether they should be hired as consultants?

Yost: Yes.

Lampson: Better not to know that. People who don't think they should be hired? We can figure it out for ourselves.

Yost: It potentially encourages —

Lampson: It encourages people to go into this business because they know they have comfortable jobs afterwards?

Yost: It encourages them to build a following and notoriety or allows them to capitalize on that and that's just an inherent wrong.

Lampson: Oh, I see. This is evil, so it's better not to know. I cannot get excited about this as an evil. I'm sorry. Raping kids, that's really bad; this is surely a matter of pragmatics as far as I can see, and I find it very difficult to believe there are very many people that are motivated to go into

the hacker business so that they can acquire notoriety which they can subsequently exploit to get a cushy job. I just don't find that scenario plausible. [Laugh.]

Yost: But once they've done it, capitalizing on it.

Lampson: Yes, other things being equal you shouldn't reward these guys. but other things are a long way from being equal. [Laughs.] Most incredible nonsense I've ever heard. I never heard this particular shibboleth before, but the whole security business is just filled with shibboleths. And you know, people tell all these lies about what are the losses to cybercrime. I don't know if you've run across any of Cormac Herley's work debunking these lies. Maybe lies is too strong [a word]; they take the numbers and they manipulate them to get the answer that they want, which is not terribly hard to do. But almost certainly, almost all that data is completely without any real foundation.

Yost: In 1992, you won the highest honor in computing, the ACM Turing Award. Can you discuss the meaning to you of winning that award; and in your Turing Lecture, you included some discussion of computer security. Can you tell me about that?

Lampson: Did I? I won that award for the Alto, because the way you win the Turing Award is by doing some particular thing that a lot of people notice.

Yost: Right. And in the description on the ACM website certainly highlights that and mentions it first, but computer security is also —

Lampson: I made some impact with the confinement paper, and even more with the access matrix. Did my previous interview tell the funny story about the access matrix? I invented that because in the late 1960's I think it was, the ACM was trying to put together recommendations for an undergraduate curriculum in computer science. So they chartered some committees. In particular, they chartered one for operating systems, and there were half a dozen luminaries on that committee. Jack Dennis was on that committee, and Nico Habermann — and I can't remember — four or five other people — and I was on it. One of the topics had to do with security, and people were invited to come in with proposals about how each of the various topics ought to be addressed in an undergraduate course. I had never really thought seriously about this before, but I chewed it over for a while. I said okay, we've had this notion of subjects and objects, which I had sort of had anyway because I was thinking about design issues, then we should have a function. You feed in the subject and the object and it says yes or no, or maybe it says read or write, or whatever. So I came to the committee with this, I thought perfectly innocent proposal, and the immediate reaction of all these admittedly young but still just fairly distinguished academics, was function? Much too complicated. We can't understand this very well, and we're sure the students won't be able to understand it. I was a little baffled by that because I had a considerable background in mathematics, which I guess none of these other people did. So I went away and came back the next day and said okay, never mind, forget what I told you before; we'll have a matrix. [Laughs.] That's something we understand. So sometimes I really wonder about our field. But I think that access matrix has had a big impact, and I'm sure that was one of the things that people were thinking about when they wrote the blurb. My own view is the only interesting thing I've done in security is the work that was done around 1990,

that we documented in a whole series of papers that Martin Abadi, and Mike Burrows, and Ted Wobber and I wrote.

Yost: Can you tell me about that research?

Lampson: We started out trying to figure out how we do distributed system security. This is an outgrowth of the work that I did with a couple of guys, Morrie Gasser, Andy Goldstein and Charlie Kaufman, from the DEC product groups. We were trying to figure out how to do security for distributed systems and in particular, for systems that span more than one organization. We wanted to be able to answer questions like, how could you set things up so it would be possible for an Intel employee to log in to a DEC system and get access to some files, if they had been properly authorized, in spite of the fact that the only credentials that the Intel employee has is some certificate signed by Intel? You need to tell a systematic story that DEC can express its trust in Intel to sign such certificates identifying John as an Intel employee, so you end up with a belief that the guy you're talking to is John at Intel. And if your data structures will permit it, you can put John at Intel on the access control list and he'll be able to get to the file. The challenge is to work out all the details of that; understand exactly what collection of individuals, organizations, programs, machines, cryptographic algorithms and so forth, are being trusted, in order to give you confidence that you're making the right decision. And how can you record all that so that it can be subsequently audited? That was the basic goal. And how can you do it efficiently enough that it could actually be successfully deployed and not bog everything down? We worked out schemes for authenticating programs too. Basically the lowest level of all these things is some piece of mathematics. You have an encryption algorithm, which essentially says

that the message is being authenticated by the key that decrypts it. And then you need some higher level stuff because you can't put keys on access control lists because they change all the time and they don't mean anything, so you wouldn't be able to look at the access control list and decide whether it's right or wrong. So you need some way of saying that that key speaks for some person or organization. Many cases, you have to go through two or three levels of that. The key might speak for you and then you might be authorized to speak for the Babbage Institute, and so on; then I could put the Babbage Institute on an access control list, and you could come in and authenticate with your key and get access. We worked out all the machinery for that.

Yost: Can you talk about its deployment?

Lampson: Subsequently, I think it was all premature when we did this work. No one was very interested in it for about 10 years, but nowadays you see these so-called federated identity systems, which basically use these ideas. And you're just starting to see some account of the identity of programs as well as the identity of people. The way I think about it, the germ of that was a paper that Dave Clark and somebody Wilson wrote, which you probably know about. Clark was at MIT and Wilson worked for some accounting firm, Arthur Andersen or one of the other Big Five accounting firms, and they were trying to figure out how to understand commercial security practice, in terms of the sort of academic security models that have been primarily motivated by the needs of the military, and they came to the conclusion that it wasn't a very good match. One of the things that was front and center in their analysis was that in many cases, a crucial part of the security system is, you've got this database that nobody has direct access to. Nobody can write SQL queries against the database itself, except maybe a few

administrators. Everyone else is accessing the database through some application, so it's the identity of the application that's crucial. They explained all this in the paper, and to some extent, they were reflecting some of the not very well formalized commercial practice. But it's become more and more clear over time, I think, that this is an essential component of practical security, because it gives you a completely flexible language for expressing what sorts of access somebody should have to some pile of data, namely whatever access the application permits. The down side of that is that you can't analyze it very well because it's a pile of code; but the up side of it is you can implement anything you can imagine in that code. The alternative approach, which many people have pursued pretty far, is to say complicated things about, well, people that are in the administrative group are allowed to access this data between 6:00 and 8:00 a.m., as long as they only touch six fields, and yaddiyaddiyadda. Very quickly these things become so complicated that you can't understand them, and besides, they don't do what you want. At least the other approach does what you want, even though, in a sense, it's so complicated that you have no hope of understanding it. But we always rely on our informal understanding of what programs are going to do, so this just seems like the only viable way of making security systems have that flexibility that they need to have in practice.

Yost: In 1995, you joined Microsoft. Can you tell me about that decision and what you worked on in the early years there?

Lampson: Yes. I was working for DEC in the Cambridge Research Lab for November/December of 1994. One of my colleagues, Dave Wecker, came into my office and closed the door. He said, "Butler, I'm planning to leave DEC and go to Microsoft. Tell me why I

shouldn't." Well, I did my fiduciary duty in trying to talk him out of it, but actually I couldn't think of any reason, because at that time it was pretty clear that DEC was not going to be able to get any use out of its research anymore. DEC was not in very good shape. It seemed pretty clear that its classical business model was not viable. So I called up my friend Charles Simonyi, who was at Microsoft, and said how about it? So I went out there and talked to a bunch of people, and they offered me a nice job. That was how I got to work for Microsoft, exploiting contacts. I originally got to know Charles when he was a Berkeley undergraduate in the late 1960s and we worked together on a whole bunch of things between 1967 and 1980, roughly. And then after that, he went off to Microsoft and I didn't have much contact with him.

Yost: There was quite a team of talented undergraduates and graduate students at Berkeley in computer science?

Lampson: Oh, yes. The team of people that worked on CAL TSS is a very impressive collection. Of the people that were in the Berkeley Computer Science Department, I think there were six professors of various grades initially, and three of them won Turing Awards. Plus, there was Jim Gray, who was a graduate student and also won a Turing Award. And then Ken Thompson, you know, worked on the GENIE Project before he went off to Bell Labs; he wrote the original ED editor for UNIX, and—this is a little bit unfair—it sort of was a clone of the QED editor that Peter Deutsch wrote and I documented for the 940 system. Many similarities.

Yost: Would you characterize computer security as a substantial portion of your work and time at Microsoft or was it other areas?

Lampson: I spent sizeable chunks of my time on various aspects of it. For example, at the time that I joined Microsoft, anti-piracy was starting to become hot and I spent a fair amount of time on that in the first two or three years that I was there. I finally gave up on that because in order to deploy anything you had to talk to the content providers and that meant you had to talk to their lawyers, and that was impossible because their lawyers drew up things that were so divorced from practical reality that you just couldn't communicate with them. So I gave up. In fact, the music industry managed to practically run itself into the ground because of the brain dead attitude that they took about this. They just could not understand the fundamental fact that you cannot use technology to protect a system unless it has behavior; if all it's doing is producing content — noise or pictures — well, it's a fundamental fact of life that if people are going to be able to perceive that stuff, gadgets are going to be able to perceive it, too. Right?

Yost: Right.

Lampson: So there's no way you can protect the music; there's no way you can protect the video; that's absolutely fundamental. Now it's true that in most of these systems there are even easier ways of doing it than listening with a microphone, but that's not very hard. So it says that preventing the easier ways of doing it is not really going to help you. You can protect games because they have behavior, and in fact, both the PlayStation and the Xbox have been pretty successful in protecting game software. They go to considerable extremes to do that. I heard an interesting talk just a year or so ago about the things that the current Xbox does differently from the previous one. It turned out that there were various breaks of the previous Xbox's security,

which basically relied on the idea that while the chip was doing its security decision, you would do unauthorized things to the power and ground pins, and this would cause the chip to behave in various undesired ways. If you did enough of that you could coax it into revealing keys, or doing something that it's not supposed to do. So the new one carefully monitors the actual physical integrity of all its signals, and it aborts whatever it's doing if it sees anything out of line. They made very substantial and very successful efforts along those lines, but for most purposes, for just ordinary things it's pointless to make these efforts because there's so many ways around it.

And that doesn't even take account of the pragmatics. If you look, for example, at how the DRM works for books that you buy for your Kindle. It's not so bad on the Kindle, but they also want you to be able to read the books on your PC, which means that, I'm sorry, but that means that the decryption keys have to be exposed on the PC. Furthermore, the PC doesn't have a nice unforgeable serial number that the Kindle has so, they can't really tell what PC is reading the book. They make some feeble efforts along those lines, but you can easily find software on the net that will crack protection, just because of the fact that they also wanted you to be able to read it on your PC. So I worked on that for a while, and I didn't do that much on security.

Yost: Did you have any involvement with the security development life cycle?

Lampson: Not really. I chatted with guys a couple times, but for the most part in the early days, they were— this is not meant to sound negative at all — they were doing basic hygiene, which we had not been doing before. Nowadays they have some fairly sophisticated technology for fuzz testing and stuff like that, but in the early days it was really just going from the wild west to

basic hygiene. I didn't have much to contribute to that. In the first half of the last decade, I did a fairly sizeable project trying to work out a comprehensive security architecture for the whole company. Basically, more than half of it was applying the ideas that we'd worked out at DEC ten years previously, and eventually it came to nothing because people basically decided they could muddle through with what they had. Security of Windows is enormously complex and there are jillions of bugs, which is kind of unavoidable, given how complicated the whole thing is and how it's evolved over several decades to meet the needs of a very wide variety of customers, for whom typically security is not a high priority. It's not that they embrace insecurity, but they don't value security very highly. So for example, I spent three hours once with an expert, getting him to explain to me how printing actually works in Windows. Turns out that there are about 12 levels of indirection between the print button in Word and the paper coming out of the printer, and every one of these levels has a security mechanism. I was sort of convinced that it was possible to configure every one of those 12 levels so that it had the security that you wanted, but the whole thing is so complex that nobody could ever do it in real life. There's just no hope. And every one of those levels is there for a reason. Somebody wanted it, and it doubtless has at least thousands, and maybe millions of clients. But from the point of view of security, it's just too complicated. Security has to be really simple or it's not going to work even if there are no bugs.

There's sort of two places where bugs can be introduced in security. One is in the code, which is what everyone emphasizes, but the other is in the configuration. The configuration, although it's less complex than the code, is much more dangerous because the code only gets written once and then modified a few times, whereas every installation does its own configuration, and typically the configuration is done by people who don't know that much about it and have lots of other

responsibilities. And so the result is that unless it's extremely simple, it's bound to be wrong. Even if it's right today, unless it's extremely simple it's going to be wrong next month, right, because an individual will come and go, new machines will come and go, and this and that will change, and whatever made sense before is going to be wrong now. It has to be extremely simple in order to avoid these problems but simple doesn't sell.

Yost: Can you tell me about the Microsoft Palladium high-assurance stack?

Lampson: Yes. This is something that actually I forgot to mention, I think this must go back into the late 1990s. Paul England had been working on the essential ideas behind what is now called the TPM, and at a fairly early stage he came to talk to me and we worked on it together. Many of those ideas we had actually thought up 10 years earlier when we were working on the Digital Distributed Security Architecture that I mentioned before. The whole secure boot mechanism was something that we had worked out in a fair amount of detail then, and then we elaborated all of the trust relationships involved in the work that we did at SRC around 1990. That stuff was all written down in a couple of papers that we published then, but Paul had some new wrinkles on it, so he and I worked on this for a while and there's a bunch of patents, some of which are up there (on my shelf). And eventually it evolved into this industry [quote] standard [unquote] TPM thing. And Microsoft started a project that was initially called Palladium, to take advantage of the TPM to build a system in which there would be a small high assurance component that would be authenticated by the TPM and isolated from the rest of the system either by a virtual machine hypervisor, or by something that didn't exist at the time but is just starting to exist now, which is

Intel's SGX mechanism. And so we did quite a bit of design on that and we wrote up some of it in a paper that was published in one of the IEEE journals; *Computer*, I think, actually.

Yost: Yes.

Lampson: And people are kind of lumbering towards that now. Something might actually exist fairly soon. And one thing that's happened recently, which we didn't fully envision, is that program verification technology has advanced enormously in the last five or 10 years and it's now possible to actually verify, from stem to stern, code that can run on bare metal, that can do interesting things like validate passwords, for example, against some encrypted database. There's a project at MSR Redmond called Ironclad, whose slogan is "Every machine instruction verified." I think things are qualitatively different than they were even just five or 10 years ago. This kind of work has been going on since the late 1970s. NSA sponsored a project called Gypsy, which you must've heard about, and I'm sure you know Donald MacKenzie's book, *Mechanizing Proof*.

Yost: Yes.

Lampson: Right. But it required real heroics to do this kind of thing in those days. Nowadays, it's getting to the point where or very close to being — and maybe even within the scope of ordinary engineering efforts — to build modest size things that are fully verified and where the verification technology is good enough that you don't have to do all the work over again after you make one small change. So I think that that work is going to actually have some payoff. The

big challenge there, I think, and it's one that hasn't gotten very much attention as far as I know, either in industry or in academia, is you're not going to be able to apply those techniques to a full-blown browser, for instance, because it's too big and it doesn't have a spec; not one that anybody can understand, anyway. Which means you're going to have to find a way to factor your application into a highly secure part and a part that's not security critical so that you can apply these techniques to the highly secure part, which isn't going to be so big. I don't think there's any totally general purpose way to do that, and we don't understand very much about how to go about it. A simple example that sort of illustrates the point is, suppose you want a high security banking application and you have this little kernel that can't do very much. One approach you can take is to tell the bank's website in your insecure browser, okay, I want to write a check for $5,000. And the bank gathers that up into a very simple piece of totally vanilla html, and passes it to the secure side. The secure side has some code that can interpret totally vanilla html — you know, no JavaScript, none of this fancy stuff — it renders the check into a bitmap, and it puts the bitmap on the screen and it says, do you like it or not? If you push yes, then it signs the order and gives it back to the insecure part, which sends the signed order off to the bank. So that's an illustration on how for that specific application, you could have a very small thing that would be highly trustworthy, that would make the whole transaction highly trustworthy, even though most of the work is being done on the insecure side. But we don't understand very well how to do this. As far as I know, only a small number of specific applications have been worked out. That's all sort of implicit in that Palladium paper. We don't emphasize that this is the big challenge, but it's there. At the time, Palladium came to nothing. You can kind of tell it was going to come to nothing because the company renamed Palladium, "Next Generation Secure Computing Base" or NGSCB for short, and nothing that's named

NGSCB is ever going to ship, right? [Laughs.] So that all came to nothing, basically because, again, there's no demand for it.

Yost: So before we conclude, are there any topics that you'd like to discuss that we haven't touched on?

Lampson: Are there any history things that we haven't touched on? I'm not sure. There was Euclid, there was all the distributed system stuff, anti-piracy. You probably know about BAN logic, for authentication. This was one of the spinoffs of the security work we were doing at SRC. It was kind of a small, self-contained thing that made quite a splash in the operating systems community. Especially since you could apply it to all kinds of authentication protocols and it was always fun because almost always they had bugs. That was sort of an off-shoot of the more general purpose work that we were doing on trying to articulate all the trust relationships that come up in distribution security. But that was one that certainly had the most immediate impact; gave rise to a thriving sub field for a while.

In 1990, the National Academies convened the first of numerous panels on computer security. This one produced a report called "Computers at Risk." I don't know if you've run across that.

Yost: Yes, I have.

Lampson: Okay. I think one could pretty much publish that report today and it would describe the current state of affairs about as well as it described the state of affairs in 1990. Very little has

changed as far as I can tell. The leading recommendation of that report — the first recommendation, not the only one by any means — is that organizations should be required to report security breaches in detail to some appropriate organization that would give practitioners and researchers access to them in some appropriate way. It's been 25 years, and that hasn't happened. The only thing that has happened is that a number of states, I think led by California, passed laws saying that if you have PII [personally identifiable information] and it gets disclosed, you have to notify the people. And that's actually had quite a bit of impact. Congress tried to pass a law like that fairly recently and failed. So you know, the most basic sort of process mechanisms for security are incredibly difficult to make happen because people understand so little about what's really important and what isn't, and they're so sensitive about this, that, and the other thing.

Yost: Donn Parker from SRI launched the I-4 group. Very large corporations, CIOs getting together and confidentially sharing, do you see something like that as a mechanism to improve things. . .

Lampson: Or a reporting system.

Yost: . . . being something that would have a significant positive impact?

Lampson: And what did they say?

Yost: Well, I've interviewed Donn Parker and asked about it and learned a few basic things, but the nature of the organization, the confidentiality requirement that incentivizes sharing, given that, it is not possible, I can't get much information about it.

Lampson: Oh, they have this thing. I see.

Yost: Yes, they launched this thing.

Lampson: But it's all top secret.

Yost: Yes.

Lampson: I see. Well that makes it less useful.

Yost: Well, less useful to the research community and those outside.

Lampson: That's right. Perhaps these CIOs find it useful.

Yost: Just by participating, continued participation, at least, it has been around for quite a while now, and so apparently participants are finding that it's worth their time to share information about their security mechanisms, infrastructure, vulnerabilities and their breaches.

Lampson: But we don't know anything about it. At what level of detail they're doing this, or specifically how many times this has caused an organization actually to do something different based on what they've learned or anything like that.

Yost: Right.

Lampson: Well that's interesting. I didn't know.

Yost: Hopefully, one day in the distant future their early records will make it to some public outlet.

Lampson: Right.

Yost: But that's probably after to be a long period of time. Some corporations have a schedule where they donate archives on a rolling basis from the distant past.

Lampson: It doesn't help us a whole lot with our security problems, unfortunately. It always seems to me that the model for this should be the way commercial airplane safety is done, where everything is basically public and there's a clear notion that you don't try to attribute blame. Not that people can't get sued for negligence, but fundamentally the system is organized to understand what went wrong and figure out what should be done to stop it happening again, rather than to affix blame, or do politics, or any of those other things that militate against the first goal. We've made zero headway on that in the computer security space. It's possible that a

consequence of using computers more and more to do critical things is going to cause us to change, at least partially. Today, I think we've made hardly any progress in that direction as far as I can tell. I'm sure you know about the Therac-25 disaster 25 years ago. From the information that I could get from my colleague, Daniel Jackson at M.I.T., things have not gotten a whole lot better in the ensuing 25 years. There's a lot of questions about what are actually effective and cost-effective mechanisms. The airline industry has its own fairly elaborate set of mechanisms for the safety and security of the computer systems that they deploy on planes, but it's not really clear that what they do is anywhere near optimal. There's a very large amount of religion associated with it. Daniel was head of a National Academies panel on how to write dependable software a few years ago. The basic story he was telling was if you were going to rely on your software in a serious way, you have to present evidence that it works. You can imagine various different kinds of evidence: appropriate kinds of testing, appropriate kinds of trial deployments, verification of parts of the code, yaddiyaddiya. You can't just wave your hands and say this looks good. You can't say, well, it's got these three features. You have to present evidence that directly bears on whether it does what it's supposed to do. This is a very tough sell, but I think as people use computers more and more for safety critical things, this may cause some change in the whole culture, because you know, it doesn't really make that much difference if 75 bits of PII are disclosed by Target; what harm is actually going to be done? You can get peoples' e-mail addresses from lots of places, and if the credit card numbers get misused the banks take a hit, and the banks and Target can fight it out who pays. There's no reason I can think of why society as a whole needs to be concerned.

Yost: It's not like a plane going down.

Lampson: That's right. But if self-driving cars drive off the road, that's more serious. [Laughs.] Or if your pacemaker gets hacked. So far this doesn't seem to be happening, but maybe it will. I think that basically what I've learned about this whole field is that people are always trying to make it more complicated and that doesn't really work. If you want it to be secure it has to be fairly simple because otherwise, you just can't figure out what's going on and the bad guy will be ahead of you. And I think that's the basic lesson. We've been fortunate that up to now, it hasn't really mattered that much.

Oh, one other tidbit that might amuse you. I think this must have been in the early 1990s—so almost 20 years ago now, could it be that long?—the National Academies had a panel studying the military's command and control system. For the most part what we learned from that panel was kind of obvious stuff about how inoperability is a problem, and backward compatibility is a problem, and procurement cycles are too long, and this boring stuff that you didn't really need to convene the panel for, you could've just made it up. But there was one thing that was a significant surprise: we took some field trips, and what we observed was unbelievably bad operational computer security in the field, much worse than best commercial practice. The question is, why? The conclusion I came to is that the responsibility for computer security doctrine in the military lies with the NSA, and the NSA's culture is the crypto culture. The crypto culture is that if it's not perfect, it's no good, and computer security is never going to be perfect. But the effect of this is that to first approximation you can't do anything, because anything you propose to do is going to have a hole. Somebody will point the hole out and the answer will be well, don't do that. So they kind of deploy it as is, and the poor sergeants who are

in charge of the gear have to struggle with it and try to figure out what to do. And they don't do a very good job, not too surprisingly. There doesn't seem to be any way around this that they've been able to figure out. Most things get screwed up because of what happens to actually deployed stuff as opposed to what the design is, because usually the design doesn't take into account practical consequences. The choice between being secure and getting the job done — people are going to choose to get the job done. Security problems are somewhere off in the future. And besides, if the adversary's really competent, you'll never even really know that you were penetrated. [Laughs.] There's this new movie about Alan Turing called "The Imitation Game" that's just come out.

Yost: I saw a trailer of that, but not the movie yet.

Lampson: All the technical stuff in that movie is nonsense, but the basic idea is perfectly sound. It was an interesting illustration of what I've often observed, which is whenever anything is in the mainstream media [and] they give details about something that I actually know about, it's always wrong, usually quite seriously wrong. [Laughs.] And this is certainly an example of that, the technical parts of this movie. But it's a pretty good movie; I recommend it. But one of its points was that the reason they were able to break the German cipher didn't have to do with the fundamental security of the mechanism — it could've been better, but if it had been used properly it would've been okay. But it wasn't.

Yost: Right.

Lampson: And that's pretty typical of all this stuff and it's very, very hard to . . .

Yost: At the system administrator level . . .

Lampson: That's right. You've got system administrators, you've got actual users. Angela Sasse at University College London does a lot of work in this area, and she wrote a good paper I guess about 10 years ago now. The title is, "The User is Not the Enemy." [Laughs.] And you know, people have studied the question of whether official rules of the organization about how you should change your passwords are sensible or not, and the answer is almost always no. If you look at what actually happens, the rules are such that it causes people to write their passwords down, or whatever, so it's all very counterproductive. People typically make these decisions in the absence of any actual evidence.

Yost: Preventing extremely simple passwords makes sense, but not making people change them.

Lampson: Right. And there are very powerful forces. Three or four years ago, the U.S. bank regulators said if a bank wants to offer online banking service, they have to have two-factor authentication. So I thought I would be getting a SecureID token in the mail. Nothing like that. You know what the two factors are? One of them is your password, what do you think the other one is? They have this deployed now, you must have encountered this if you do online banking.

Yost: What is the second?

Lampson: It's the device. The first time you use a new computer, they ask you a bunch of security questions, and then they leave a cookie, okay? So that's the second factor, and they got that past the regulators. For all I know, that's perfectly rational. I don't know what the statistics are for fraud from online banking, but my guess is it's not as simple as you might think to get significant amounts of money out of the banking system.

Yost: It can't be terribly easy.

Lampson: Or they would be doing something different. I was amused to learn from my T.A. last spring that in Norway, if you want to do online banking, the bank sends you a little card the size of a credit card. On it are 20 or 30 scratch-off six-digit codes, and when you log onto your account it says use number 23, and you scratch off the code for number 23 and type it in. So it's one time passwords, and after you have only 10 left, they send you another card in the mail. I believe the current prevailing wisdom is that U.S. consumers wouldn't tolerate this. It just shows that cultural factors can be significant, but it also goes to show that you don't need high tech to do things significantly differently than we do them today. But all this stuff doesn't get much attention and people like to — I'm sure almost all the people you've talked to focused their attention on other places that are probably much less important for security.

Yost: Yes, for the most part.

Lampson: And I've done plenty of that myself.

Yost: With it being a specialization in computer science, I think that there's some graduate students that are doing research on very practical things, but many of them aren't. And I think that there isn't great attention paid to real world problems, as you point out, and focusing on practices that truly make a difference.

Lampson: I'm afraid that's right. Microsoft has been observing this since it's tried to build up its cloud infrastructure, which sort of started with existing Windows server stuff. Minor questions like, if you have a data center with 200,000 machines in it, how many administrators are there that are in a position to wreck it? Well, it could be very few if you're engineered properly, but if you just straightforwardly deploy the software that was built for data centers with 50 machines, the answer's going to be a lot. So there's all kind of things you have to learn in order to operate in a sensible way at scale. We at least have been learning them very slowly and painfully. [Laughs.] Okay, well, if you have other questions that's fine, but otherwise I think I'm played out.

Yost: Thank you very much. This has been very helpful.