An Interview with

PETER G. NEUMANN

OH 425

Conducted by Jeffrey R. Yost

on

3 June 2013

Computer Security History Project

Menlo Park, California

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

Peter G. Neumann Interview

3 June 2013
(Final editing 6 December 2013)

Oral History 425

Abstract

In this interview, computer security pioneer Peter G. Neumann relates his education at Harvard University (A.B. in Math, S.M. and Ph.D. in Applied Math), including an influential (to his perspective and career) two-hour long meeting/discussion as an undergraduate with Albert Einstein (discussing "complexity" and other topics). The vast majority of the interview addresses the many facets of his highly influential career in computer security research. With regard to the latter, this includes discussion of his work at Bell Labs and extensive involvement with MULTICS security, and his subsequent four-decade (and continuing) career as a research scientist at SRI International. He tells of his work and leadership with the Provably Secure Operating System (PSOS), research and writing on risks (including moderating the ACM Risks Forum), insider misuse and intrusion-detection systems (IDES, NIDES, EMERALD), and his current work on two DARPA-funded projects that builds on key lessons of the past to design and develop secure/trustworthy computer systems. He also relates the computer security research infrastructure and how it evolved, as well as comments on a number of other topics such as the major computer security conferences and the range of perspectives of researchers in the computer security research community.

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 University of Minnesota's Charles Babbage Institute, and I'm here today at SRI International with Dr. Peter G. Neumann. Peter, can you begin by answering a few biographical questions — when you were born and where did you grow up?

Neumann: Sure. I was born in New York City in September 1932. My parents and I lived in Manhattan for a year and a half, after which we moved out to the suburbs in Rye, New York in 1934. I went through the public school system in Rye, which at the time was one of the best high schools in the state. I had a wonderful high-school experience with lots of science and math and music, and everything else, sports.

Yost: What were your favorite subjects in high school?

Neumann: Favorite subjects: I was very eclectic. I was head of the civics group and we used to go to the UN and hang out there and listen to delegates talk. I loved math and English. I had an English teacher who is memorialized on my website, Marsden V. Dillenbeck. I wrote a limerick with two sets of annotations over multiple years elaborating on the text. MVD was someone who really inspired me to write proper English and poetry, and to be a literate person; he had a tremendous impact on my life. Of course, music was also very important, as were math and science.

Yost: At that time of your life, can you think of any other individuals who were extremely influential to you?

Neumann: In my high school days?

Yost: Yes.

Neumann: Primarily the math and music teachers were the real influences, along with my English teacher – who was phenomenal, absolutely inspiring -- and many others. I had a Latin teacher who was an absolute delight. She had a bust of Caesar on her desk, and was very much living in the world of old Latin culture. I actually had a letter published in Latin, in a Latin school magazine. I was very much enjoying everything I did in high school.

Yost: And you did your undergraduate work at Harvard?

Neumann: Undergrad and graduate at Harvard; Undergraduate in math, and graduate in applied math, before the existence of computer science.

Yost: When you started, did you know you wanted to major in those subjects?

Neumann: Well, I always loved math, and I went into Harvard as a math major. I took Howard Aiken's computer architecture design switching theory class in the spring of 1953. And then suddenly I found myself getting an internship in Washington DC (actually, Naval Ordnance Lab in White Oak MD) for the summer of 1953, after my

junior year, which led me to program a very primitive computer -- writing a complex matrix inversion program on a plugboard microprogrammed punch-card machine (the Card Programmed Calculator). But the 1953 class with Aiken was a graduate course, and I wound up having at least five graduate courses by the time I finished my undergraduate degree, all in computer-related topics. And then four years of graduate school at Harvard.

Yost: Other than Aiken's course, do you remember some of the others?

Neumann: Oh, I remember vividly both professors and colleagues. We actually have an annual dinner each June with a bunch of my old graduate colleagues. We do that once a year as I'm heading off to my old music camp reunion. But the professors, other than Aiken, were Tony Oettinger, who eventually became my graduate advisor; Bob Ashenhurst, who was on the faculty after he got his Ph.D.; Bob Minnick, whom I really enjoyed; Ken Iverson of APL fame was my thesis advisor for a couple of weeks before he decided to go to IBM; and lots of others. My graduate colleagues whom I'm still close to include Albert Hopkins, who later was co-designer of the Apollo Guidance Computer when he was at Draper Lab. Albert and I, and Fred Brooks and Bill Wright collaborated on a paper that was published in 1957, on composing music on the Harvard Mark IV. It turns out that Bill and Fred had taken a seminar with Tony Oettinger on statistical linguistics. They had done a Markov chain analysis of the sequence of notes in 37 common-meter hymn tunes. So, the following year in Oettinger's seminar, Al Hopkins and I generated 636 `new' hymn tunes based on the statistical sampling of the 37 hymn tunes using the multigram Markov probabilities ranging from one to eight successive

quarter notes. We effectively `composed' hymn tunes, from purely random up to statistically hymn-like. Perhaps the most interesting thing was that when we spanned about four consecutive quarter notes, we started producing hymn tunes that sounded very 'hymnish.' When we went to our maximum length of eight successive quarter notes in the Markov chain, we wound up with hymn tunes that sounded like real hymn tunes. There was even one extreme case in which the entire first half of the generated hymn was composed of corresponding parts of two different hymns from the sample space conjoined, and the second half was an identical copy from one of the sample tunes. So it sort of converged as we reached the maximum length of our Markov chain. This was a lovely experiment in statistical linguistics and the structure of music, and sort of the complexity of how musical sequences are constructed. In addition to Fred and Al and Bill Wright, other memorable grad-school colleagues included Marty Cohn, who became a professor at Brandeis, and did a lot of work on compression coding; Bill Eastman; Jim Lincoln, who worked with Al Hopkins at the Draper Lab on the Apollo and other very reliable computers. Peter Calingaert eventually went to work with Fred Brooks at North Carolina, after their years at IBM. Robin Esch became a professor at Boston University in the math department. I have wonderful memories of those days and, of course, I was as eclectic as ever. As an undergraduate I had played in the Harvard-Radcliffe Orchestra, and performed and conducted Gilbert and Sullivan, and sang with the Harvard-Radcliffe choral groups and the Boston Symphony, including on the RCA recording of Berlioz's Damnation of Faust. I was one of the leads in the world's first science fiction opera in my first year of graduate school. And then the second year, I co-founded the Harvard Opera Guild, for which we did performances of *The Barber of Seville*; I played bassoon in the

pit orchestra, which I managed. We picked up most of the players from Mike Senturia's Bach Society Orchestra, and one night when one of the minor leads couldn't make it, I performed his part instead. I had a very well-rounded time at Harvard, with lots of music, squash, tennis, basketball, and softball. Very rounded existence.

Yost: A wonderful time intellectually.

Neumann: Absolutely. Somebody who was a computer nerd — was not me — at that age I was doing a lot of other things as well.

Yost: You briefly touched upon it, but can you talk a bit more about your 1953 summer job where you worked on the IBM card-punch program?

Neumann: Yes. The program I wrote was punched into a card deck almost two feet long: a fairly complicated program for a complex matrix inversion. I put the data deck at the back, and ran it through; it punched out two cards, which went to the back of the increasingly large deck and ran through again, punching out four new cards. This kept up until it finished. And then the matrix and its inverse were multiplied together to check the precision of the results. I grew up with Aiken's belief that you always checked your results in some way. In fact, one of the cute stories of Aiken's lab was that he always had women working on hand calculators in the basement of the Computation Lab, differencing the results of the tables that had been produced by the Harvard Mark I. For many years, these ladies never found any problems; all of the differences eventually

reduced to zeros, so everything was fine. On one of the later sets of tables of perhaps Bessel functions, suddenly they discovered a systematic error. And sure enough, those results were in error. They traced it back to the program, and discovered that this was the first program in which the highest numbered register had been used as an adder. It had been used for storage correctly in the past, but never as an adder. This was a 23-decimal-digit rotating step-switch register where the input wires had been crossed on the least two significant digits. But they were also crossed on output, so when you used it as storage, it gave the correct result. On the other hand, when you added, the carries went askew. This hardware flaw had never previously been detected, and was found only because of Aiken's insistence that the results had to be checked. So, when I got to write my complex matrix inversion, of course I multiplied it back together again to see how close the results were to what they should be. This was a lovely exercise in writing a very complex program for a machine that had basically nothing that today we would call memory. It had four registers, and everything else was on the cards. So you're quite limited to no program loops, although the programming language was something that had been created via a plug board on a very primitive machine that had been rigged up to be a 3-address machine. It was something like a hardware-based compiler. When I got to the Harvard Mark IV, as I did in 1953-1956, the Harvard Mark IV had an amazing compiler that was actually built in hardware. You used a symbolic programming language. You entered your program onto a machine, which prepared a tape that you read into the Harvard Mark IV memory, and the quasi-compiler or assembler converted the symbolic notation into a machine-code program -- there's an example of a kind of hardware-based program compiler. Thus, I go back to extreme primitivism in the Harvard Mark I and the Harvard

Mark IV, and the Card Programmed Calculator. And then along came the Univac I. There was a story that came out about how it had been the Univac One—well, actually, it was in a poem written by David McCord, who was the Poet Laureate of Harvard, not the national poet laureate:

> O God, Our help in ages past, Thy help we now eschew.
> Hymn tunes on Univac at last, Dear God, for Thee, for You.
> We turn them out almighty fast, Ten books to every pew.

In fact it had been the Mark IV. So again, we have a few errors in history that sometimes get corrected, sometimes don't. There's another one that I might mention; Peter Salus's book on UNIX attributes to me the name of Unics [UNIX]. Brian Kernighan is probably the person who actually named it Unics, but I'm not absolutely sure, and he's not absolutely sure. But I certainly was observing that the very first version was an emasculated one-user Multics that Ken Thompson had created before it morphed into what became Unix – so that the name of Unics seemed very appropriate at the time (except to the AT&T PR folks, who presumably requested that it be changed to Unix). Okay, so that takes us through some of the early days.

Yost: I understand that you had the incredible opportunity to have a meeting and discussion with Einstein in your youth.

Neumann: Yes. This is an amazing story. In front of you is a photo — actually my son took one of my mother's slides and printed it backwards. The transparency below it is the correct face. (A photo is on my website.) Einstein's stepdaughter, Margot, came to mother in 1945 and asked, Will you teach me how to do mosaic? My mother was one of the three best-known mosaicists in the country. And my mother said 'of course, and I've always wanted to do a portrait of Albert, so is there any way we can arrange that?' My mother used to sit at his feet in his study in his home on Mercer Street, in Princeton. She would sit motionless for two or three hours while he was thinking and occasionally writing notes. Each time, she'd go home and from memory draw what became several pastels. Then she took a year off and did the mosaic portrait. When I was going to be singing in the Harvard Glee Club at Princeton in November 1952, my mother said why don't you call Helen Dukas, and maybe you'd get to see him for five minutes or something like that. Before the concert Friday evening, I called Mrs. Dukas. She said, He absolutely adores your mother. Will you come by for breakfast? And so it was that I had more than two hours of fantastic intellectual stimulation discussing complexity in mathematics, chemistry, physics, cosmology, and the universe. What really got me fascinated was complexity in music, which of course tied in with everything I'd been interested in, in the evolution of classical music. We started with Gregorian chant (a single line and modal, and purely consistent with whatever mode it was in). And then Bach, and fugues, and Mozart, and Beethoven, and it gets more and more complex. So I asked him, tell me, what do you think of Brahms? And he sort of grimaced and said, "I have never understood Brahms. I believe he was burning ze midnight oil, trying to be complicated." And that really, really struck me because his whole concept that everything

should be made as simple as possible, *but no simpler*, was sort of the essence of what we were talking about. We initially talked briefly about the U.N., politics, Israel, and general stuff. But the real essence concerned complexity in one form or another. And this notion that Brahms was too complicated for him was fascinating to me because I love Brahms, I love the inner voices, and the cross rhythms, the two against three, and three against four, and all sorts of intricacies that Brahms created. And so to me, it was wonderful to hear that Einstein did not like what he perceived to be the excess complexity of Brahms. Now, I've always wondered what he would think of some of the Philip Glass-like minimalist music of today, or some of the very atonal stuff that happened in the latter part of his life. He probably would've hated that. On the other hand, some of it is simple and structured. I mean, if you think of Philip Glass's music, it is minimalist in a very real sense. So Einstein might've just liked it, even though it was different from the classical stuff that he seemed to enjoy most. We also talked about his string quartet and a little bit of piano playing. All of this was truly delightful for a kid who was a junior at Harvard, having taken chemistry, physics, and a lot of math, humanities and English writing, and so on. This was an eye opener to have a free-ranging discussion somebody who really had thought so deeply about so many different things. He was definitely not a one-trick pony; indeed, he was an amazingly deep and thoughtful person, so it was just a tremendous joy for me. And the interesting thing, I think, is that this discussion had a tremendous influence on essentially all of my research. In Multics in 1965 at MIT, working from Bell Labs, complexity was one of the key issues. How do you structure a very large system and have some assurance that it would be secure and reliable? Part of the answer was new hardware that separates things—segmentation and paging in hardware, and

abstraction, composability, and encapsulation in software. The system layers of abstraction and the layers of complexity were very cleanly delineated in Multics. And then the ring structured layering in the operating system, and hierarchical directories, and dynamic linking in the symbolic naming of files, and later, when Ken Thompson joined us, of input-output streams as well.

Yost: A lot of the security worked into the original design of Multics.

Neumann: Yes. The security design was proactive, in the sense that Ted Glaser and John Couleur had developed baseline hardware that would enable Multics to be much more secure than any other system, including the IBM 360-67 — which really didn't understand the essence of segmentation. To the IBM folks, segmentation was not a set of independent entities, it was just a linear virtual address space that happened to be divided up into equal-sized chunks, rather than independent segments. Similarly, when we get to PSOS, the Provably Secure Operating System, you'll see that the layered abstraction hierarchy is very fundamental; the big innovation there was designing the hardware and the software in a common formally specified language. When we get to talk about what we're doing today, where we're building new hardware and new software, all of this stuff from Einstein's notion of complexity, that it should be as simple as possible but no simpler, strikes home again because we are specifying the hardware in a high-level language that can be directly compiled into hardware. Once again, the layers of abstraction become absolutely fundamental. Each PSOS layer was conceptually simple, which is something we really tried to achieve in the design of the secure operating system

and its hardware, and each layer was completely specified in terms of what needs to be done. The mappings from layer to layer were completely specified. The abstract implementations of how one would implement an abstract program at one layer in terms of the lower layers were completely specified with all of the exception conditions and preconditions and post conditions, and so on. So this was a dramatic step forward in terms of being able to deal with complexity. Again, it all goes back to my discussion with Einstein, which really set me thinking about how do you deal with very complex systems without oversimplifying? And that was really the essence of Multics, and PSOS, and the work that we're doing today with DARPA.

Yost: Did that discussion with Einstein involve any talk about computers and complexity, of designing architectures or operating systems?

Neumann: Not at all. We were just talking about complexity, and I think discussing complexity in music was one of the things that he particularly enjoyed because he was very knowledgeable musically. He was a good second violinist in a string quartet in Princeton, and he knew the repertory. He knew the string quartets; he knew the symphonies; he knew a great deal about music, and it was clear that he had thought deeply about complexity, not only in cosmology and physics and the universe, but also in things like music and in life, in some sense. The idea that you should make your life as simple as possible but no simpler is also relevant to the way you conduct your daily existence. I asked him about how he could concentrate for hours on end and never move (as he had done when my mother had visited him); did he ever get blocked completely,

where he felt he was simply not able to think about the problem? He said oh yes, many times. He said he'd get up and walk in the woods or play a little music or do something else, and then it would come to him without even thinking about it. So he learned sort of the art of Zen, doing without doing, or *wei wu wei* in Chinese, as in the *Tao te Ching*. The idea is that you don't force something, you just let it come to you naturally. I learned so many things from that one discussion with him.

Yost: I understand you used the Harvard Mark IV in your undergraduate thesis. Can you talk about that?

Neumann: My ad-hoc professor in Applied Math (rather than my literal professor in the math department) was Philippe LeCorbeiller (whose son was an editor of *Scientific American),* and he was a wonderful human being. He was the one who graded the French and the German exams that I had to pass for my master's degree. He was my inspiration for the senior thesis, which was on elliptic integrals. The thesis was essentially something that nobody would ever do today, involving a nomographic representation of five different types of motions that could be described by elliptic intervals. I generated tables on the Mark IV as part of Ken Iverson's course that I took in my senior year and actually wrote that program with a graduate student, Ralph Kilb. This was an interesting story by itself. Ken Iverson wanted us to demonstrate our debugging skills, and we were (perhaps arrogantly) convinced that it would run. [Laughs.] And Ken said, oh no, nobody can do that the first time. The Harvard Mark IV was very inflexible in many ways, as you had to actually plant a program checkpoint manually. Then when that location in the program

was next reached, the contents of a register could be displayed in the digital wheels above the machine. We did this for about 30 minutes, and everything that we said was going to happen had happened exactly. This was basically the first time we'd actually gotten access to the machine under real conditions. Finally Ken said okay, maybe I believe you, so let it run. And it ran, and printed out a set of tables, which are in my undergraduate thesis. But what I learned from that was that it pays to do structured walkthroughs, if you will. This was a thousand-line program. And it was, again, quite complex -- but Ralph and I had convinced ourselves that it was correct, after intense manual examination, which seemed unlikely at the time. The Harvard Mark IV had a very clean programming language at the time. But getting back to the thesis, the inspiration of LeCorbeiller on that was magnificent. He really led me through what I needed to do for the thesis, and it turned out to be a lovely opportunity to actually get into the computer world even though I was still a green undergraduate. That was a much more serious programming exercise than what I'd done in the summer of 1953. So that was the academic year 1953-1954.

Yost: Did you go straight to graduate school?

Neumann: As I said, I'd already taken five graduate courses by the time I finished my undergraduate years. Aiken wanted me to help write the book on the Harvard Mark IV, which turned out to be a much more complicated thing than I was able to handle as a grad student who was working part time as a research assistant. But those graduate years were wonderful in the sense that there was an open-ended Bell Labs research grant to Harvard, and I wound up working on that, writing a bunch of Bell Labs Report sections. My thesis

sort of evolved from that into survivable communication systems, which would tolerate arbitrary errors even better than a byzantine system; no matter how badly it fails, it's going to resynchronize itself quickly, even though these are variable-length coding schemes with no overt synchronization

Yost: How did you come to that topic?

Neumann: Well, I was interested in statistical linguistics already, even before Tony Oettinger's seminar. Oettinger had been influential in my thinking — but I was very interested in compression coding, and I was very interested in the effects of hardware and software errors on systems. So I did some work on error correcting codes in my early years and some of these self-resynchronizing sequential coding schemes. I have a lovely story — when I first got to Bell Labs, I really enjoyed Huffman's paper on Information-lossless Sequential Machines -- that is, sequential machines that could run backwards or forwards, albeit backwards with some delay. If you had the right initial state, you could actually run the thing backwards. So I wrote a paper, which I actually submitted to what was then the IRE, on some of Huffman's automata that were in fact self-resynchronizing in the presence of hardware errors, *in both directions*. I then had a phone call one day from Dave Huffman saying hey, this is one of the most interesting papers I've ever read. [Laughs.] Will you come up from Bell Labs and discuss it with me at M.I.T.? This was in 1963. So I hopped on up, and got to know Huffman really well. He was another one of my inspirational mentors; somebody whom I found to be absolutely fascinating. On my laptop I can show you some of the foldings that he had created, which are marvelous. He

had written a paper on the continuous deformation of semi-rigid surfaces, and used that to develop ‘foldings’ like Origami but with smooth curves that are phenomenal. You might look at it and say there’s no way anyone could develop that out of a flat surface, and yet Huffman was able to do that because he’d done a deep analysis of the mathematics underlying what happens at each point in this surface. There’s also some of his highly intuitive work on graph-based error-correcting codes, where the minimum distance of the code is the minimum loop length of a nondirected graph. And this research was also gorgeous. I don’t know that he ever published it, although I have a set of mimeographed notes that he wrote. In any event, it’s beautiful work, and I have included some of it on my website for a course I taught at Maryland on survivable systems in 2000. So, survivability was an issue to me in my graduate school years, survivable communication systems in the presence of arbitrary noise or disruption. This is another thread that runs through a lot of error-correcting codes and fault tolerance, a report we wrote for ARPA in 1973, and the report I did for the Army Research Lab on survivable systems and networks in 2000. And that was really, again, relating to the Einstein discussion of how we might build complex systems that are reliable, secure, highly available, survivable, and all of these things together at the same time. Some people might say, well we’re going to solve the reliability problem, but we don’t worry about the security problem -- or vice versa. Interestingly, three of my former colleagues here at SRI, Les Lamport, Rob Shostak, and Marshall Pease, have just been awarded the Laprie Award for the work they did here on byzantine agreement, which was seminal to the whole field of ultra-reliable fault-tolerant computing in the presence of arbitrary failures. So that sort of ties in with again this thread of very survivable systems, as in my Ph.D. thesis, on survivable

communications in the presence of arbitrary noise and yet, very efficient variable-length coding that could result.

Yost: Tony Oettinger was a long time friend of our Institute, so I had the pleasure of meeting him, a very gifted and generous person.

Neumann: Good. Wonderful person. I try to meet with him when I am in Cambridge each June. I didn't actually finish my Harvard thesis until after I came back from Darmstadt, but my first year in Darmstadt I completed the writing of most of the thesis. However, when I got back, Tony said you need another chapter on how does this relate to some of the Chomsky phrase-structured grammars, and things like that? As a consequence, I did not finish my Harvard thesis until April 1961, when I was at Bell Labs. I was at Darmstadt, on a Fulbright from 1958 to 1960, with my professor there being Alwin Walther, another remarkable person who had a wonderful influence in my life. Toward the end of my first year, he said why don't you stay another year. I noted that the renewal deadline for Fulbrights was already two months past. He said don't worry, I have friends in Bad Godesberg. I'm going to take care of this for you. You can teach a course for me, you can write a doctoral thesis (in German), and you can help me build the computer curriculum for the future of Darmstadt. So, I stayed for a second year. Intriguingly, Darmstadt has had a tremendous renaissance in the past few years. David Parnas was there in the 1970s, after his years at Carnegie Mellon, and now there is a new Fraunhofer Institute for computer systems that has been built in Darmstadt. Darmstadt has once again become a major player in computer science in Europe. Of course, I'm

now working with a superb team at the University of Cambridge, and we'll get to that later. I presume that you're sort of following this chronologically. I also had a lot of involvement with Brian Randell at Newcastle, along the way, many years ago.

Yost: Another terrific person, I have had the pleasure of serving on the *IEEE Annals* editorial board with him for a number of years.

Neumann: When I was visiting Newcastle the first time, Jim Horning was there. He had two visiting appointments, so I got to know Jim, and John Rushby, and other folks at Newcastle. That was particularly interesting, because the Newcastle research was sort of running parallel to some of the research I was doing; they did a lot of work on complexity and security and reliability. There's also a paper by Butler Lampson that's important here. Lampson wrote a paper on hardware for the 1974 IFIPS, which said in effect that if it's not secure it's not reliable, and if it's not reliable it's not secure. Which again, is seminal in the sense that if you're trying to solve one of the problems and not the other, you're falling short. For example, in the byzantine agreement algorithm, at most k out of 3k+1 components can be arbitrarily misbehaving, maliciously or accidentally. This solves both some of the security problems and the reliability problems, except that if you can compromise more than k components (or indeed all of them), then it doesn't address that kind of a correlated attack. But the idea that you can build a system that is able to withstand serious attacks is something that again came up in my survivability study for the Army Research Lab, where I list 22 ways that you can increase trustworthiness despite the fact that you're dealing with systems and subsystems that are not so

trustworthy. Cryptography is an obvious example, where you're taking an untrustworthy communication line and making something that is much more trustworthy, if you can assume that the key management hasn't been compromised and that the key hasn't been compromised – and that you don't have denials of service attacking the communications medium. Similarly, error-correcting codes give you an amplification of the reliability of the medium but, again, you've got to assume that you're not compromised on both ends if you're trying to do security, and that the errors are not exceeding the strength of the code. I looked at a whole bunch of ways in which you could in fact increase the trustworthiness through structural or architectural approaches. This all fits together in a world view that says, if you don't worry just about one thing at a time, you're really trying to solve a more difficult problem -- which in extreme is trustworthiness that encompasses (for example) security, reliability, availability, interoperability, survivability, predictable evolvability, and lots and lots of other things. If you have a highly structured system, then you have a much better chance of being able to solve multiple problems. One of the examples there was Multics. The effort in the 1970s to retrofit multilevel security into Multics turned out to be relatively easy to do because Multics was so well structured in the first place. But maybe we'll get to Multics further on.

Yost: Can you talk about that transition from completing graduate school to working at Bell Labs?

Neumann: Yes. Before I went off to Darmstadt for my Fulbright, I had two offers. I had one from IBM Research Lab; before it was at Yorktown, it was at the Robert S. Lamb Estate — a converted property that IBM used to house a few researchers while they were building Yorktown. And I had an offer from Bell Labs. They were both really wonderfully appealing opportunities. I wound up accepting the Bell Labs offer, which they graciously extended twice. So the transition was relatively straightforward. I came back from Darmstadt — there's one thing I want to mention before going into the transition. When I was going over to Darmstadt, my former office mate, Rick Gould, was going to meet me when the boat arrived in Germany, because he was about to go back to the U.S. and accept a job with Dick Shuey at the General Electric Research Lab in Schenectady. As it turned out, in April or May, Rick died tragically in an ice slide on Dent Blanc in Switzerland, which was deeply upsetting. Rick was a good friend in graduate school. I should have mentioned him previously, because he's somebody that I remember very well. It was really terrible that he had suddenly disappeared into a huge ice crevasse, climbing together with a very experienced British friend.

I returned to the U.S. to work at Bell Labs in 1960. In the first few months I was getting settled, and in my spare time finishing up my Harvard thesis. I had Dave Farber as an office mate for at least five days, when he first came to the Labs. My first supervisor was John Runyon, who was about six-foot-six or -seven and his job was trying to detect the blue-box phone freakers and other telephone system penetrations. He was trying to anticipate all of the things that they could use against the telephone system, and consider what the Labs needed to do to overcome those threats. He was a very delightful person,

and I enjoyed my interactions with him enormously. He was there only for a short period, and he seemed to have gone off to do other things pretty quickly. My lab director at Bell Labs was Tom Crowley. He was a very warm and friendly person, and great to work for. His wife had 11 children in 11 years and the 11th was born the same year and time that my daughter was born. So I asked Tom, what is the point at which you would like to stop? He's a good Catholic of course. And he rather sheepishly says, five? [Laughs.] He and Rita had 14 kids in 15 years, which is rather amazing.

I kept working on error-correcting coding and survivable communications. I did a paper on Gilbert error-correcting codes, and later, actually, when I was at SRI, I wrote one on arithmetic-error-correcting codes with T.R.N. Rao. That was a lovely little piece of mathematics involved in being able to detect errors in an arithmetic unit. But my first few years at Bell Labs were basically continuing the kinds of things I'd been doing in my Harvard research. And then one morning, I was walking in from the parking lot (it's the day after New Year's) talking with Vic Vyssotsky (who died recently). I said, what are you up to? Oh, he said, we're having a first meeting today on what we're going to do with M.I.T. (on what would become Multics). He said, why don't you come along? You might be interested.

Yost: What year was this?

Neumann: January 1965. So, I said that'd be great and I did. After Vic went over to Whippany to work on anti-ballistic missile defense, I wound up running Bell Labs' Multics effort, commuting to M.I.T. almost every other week — with Corbató, Glaser,

Jerry Saltzer, and all the Multicians at M.I.T. and Honeywell; and Vyssotsky (initially), and Joe Ossanna who did a lot of the I/O stuff at Bell Labs, and Doug McIlroy, who later became my lab boss after Tom Crowley went off to Whippany as well. Bob Morris, who was a classmate at Harvard, was also somebody I knew well from Bell Labs.

Yost: While you were at Harvard did you have any interaction with the M.I.T. computer science folks?

Neumann: Huffman came over and gave a talk for a Harvard conference in 1955. It was a lovely result, so I got to know Huffman just a little then. But really, it was 1963 when we really interacted, and 1964 when he invited me to visit him at Stanford.

Yost: Were you aware of CTSS? Had you used that?

Neumann: Oh yes. I was aware of it but it wasn't until the Multics effort when I had a CTSS account and was using it quite heavily. The Bell Labs years were phenomenal. Then Ken Thompson joined our lab in probably end of 1967 or beginning 1968. Stu Feldman was a summer student when he was perhaps 16 – and he was already a graduate student in aerospace at M.I.T. He had sailed through college in short order. [Laughs.] Stu was around pretty much every summer for a while. What a wonderful collection of people! Those years, my vice president was Ed David, who became Nixon's science advisor. During the Multics years, I reported to my computer science lab, but I also reported directly to Ed David for Multics. Ed David was just a delight to work for. Again,

I've lucked out having had people who were just absolutely wonderful. Tom Crowley, and Doug McIlroy, and Ed David, at Bell Labs were all just marvelous as mentors, if you will.

Yost: I understand you and Bob Daley developed the Multics file system.

Neumann: Yes. Well, what happened was that on Memorial Day of 1965 we all got together — the M.I.T. folks, the Honeywell folks, the Bell Labs folks — at an AT&T training center in Hopewell, New Jersey, for two weeks basically as a retreat. Bob Daley and I wound up responsible for the file system, and developed the multilevel tree-structured directories, access-control lists, links, and backup.

Jerry Saltzer took on much of the scheduling and structure of the operating system; and Bob Graham wound up with the linkage system. Ted Glaser was an enormously influential mentor to me. He and I wrote the first set of Multics system development principles. These early design principles became the first section of the System Programmer's Manual. Although they were somewhat superficial, they perhaps inspired the Saltzer-Schroeder paper, where you have a wonderful carefully stated set of security principles. Working with Ted Glaser was an eye opener. He was blind, but had incredible long-term vision. One of my absolute favorite stories was in this Hopewell, New Jersey retreat, in a room with windows on one side and three sets of blackboards all the way around the rest of the room. We started out at nine o'clock the first morning, next to the window, going clockwise around the room, writing things in collaborative effort, talking,

writing, erasing, and so on. Around four o'clock in the afternoon, we were filling up the third blackboard, all the way back toward the window on the other side of the room. Somebody proposed something, and Ted Glaser said no, no, this contradicts what we had agreed on somewhere around 9:30 that morning — and he pointed to what was still written on the blackboard inside a rectangular box (where everything else around it had been overwritten or erased several times over). He was blind, remember?

Yost: Listening to what's being said and recalling the location of the blackboard writings [pause]

Neumann: Yes. Ted Glaser was someone who would stand talking with you at a conference reception, holding a glass. He didn't drink, so he'd have a glass of water; by the end of the reception, he'd have a detailed knowledge of everything that companies were planning to release in the future, from listening to five conversations simultaneously. While he was talking with you, and drinking a little bit of his beverage, he was able to hear and understand four or five conversations simultaneously. He was able to play tapes at two or three times speed with a compandor. Thus, he heard speech at the speaker's voice frequency, and he would memorize computer manuals when played at high speed; I don't know what his top speed was. He was a phenomenal human being with, as I said, infinite sight despite the fact that he was blind. There's one other story that is worth telling that I've almost never told. There was one summer when I was on my vacation up on the Martha's Vineyard, and we had to go down to Bell Labs for a meeting, instead of everybody else coming up to M.I.T. We did that periodically. So I

flew to Boston, met Ted at the airport, and we flew down to Newark. I rented a car, drove us to Bell Labs, and later drove us back to the airport. On the way back to the airport, somebody driving beside us honked his horn and pointed to the right rear wheel of the rental car. So I pulled over and discovered that the lugs were loose on the wheel. So Ted, in his infinite wisdom says, while you're at it, check the other three wheels. Okay. It turned out all four sets of wheel lugs had been loosened. So I tightened them up, drove back to the airport to the Enterprise Rent-A-Car desk. At the time, they had a booth in the middle of the airport. We were ready to complain that somebody had tampered with our vehicle, and the customer on the other side of the booth was complaining that the entire interior of his car had collapsed on him while he was driving at speed on the highway. When we finally got to talk to the agent, he admitted that they had recently fired somebody who was well known to everybody in the crew, and they apparently had failed to terminate him in a way that — good security lesson here — enforced complete termination. When you delete something, you delete it properly. So this man came back into the lot, and apparently sabotaged a bunch of cars. Again, Ted, in his infinite wisdom had said, check the other three wheels. He was clearly someone with tremendous foresight and experience. One day when he was working at IBM, T.J. Watson called him in and said, if you're going to work at IBM, you have to wear a hat. And Ted said, I'm sorry, sir, but I cannot wear a hat because it distorts my senses; when I walk down the hall, I can hear echoes off the walls, and if I am wearing a hat, I am likely to run into the walls, and I would be very unhappy. Watson said no, no, you have to wear a hat. Ted persisted and finally got Watson to relent. Ted was an extraordinary person. He played the organ, and had earphones so he could play at night without waking up anybody in the

neighborhood. He was very musical as well. He'd worked for Burroughs and NSA. He became the chairman of the department at Case Western. I visited him out there once. Corby – Fernando Corbató -- is also a phenomenal person. I'm still very close to the Corbatós, and my wife and I visit with them at least once a year. Jerry Saltzer has been part of my ACM committee since I took it over in 1985. There are many friends from the MIT and Bell Labs years with whom I'm still in touch with, such as Tom Van Vleck. Just an extraordinary bunch of folks I've had the honor, privilege, and pleasure to work with over the years.

Yost: You talked a bit about the division of labor in the security design principles with Multics. Can you talk a bit more about that and what role Corbató had?

Neumann: Corbató, of course, was the real intellectual head of the whole thing. He'd had his experience with CTSS. Bob Daley was one of the key guys in the CTSS years. CTSS was a very limited system — upper case only, six-character file names, flat file system, single-level directory, no hierarchy — and a lot of limitations. It was primitive in the sense that it was running on a 704, or a 7090 eventually, and nobody had ever built a time-sharing system. There were only two such systems in the early days: Dartmouth and MIT. And the MIT one was basically a bunch of students and a little bit of staff writing some code, and developing a time-sharing system. One of the most primitive things was the context editor. We're not dealing with any visual screen stuff, this is all printout — teletype or whatever. The CTSS editor had the problem in that it used a fixed pair of file names for the temporary files, but the person who designed it never anticipated that two

different people might be editing different files at the same time in the same directory. As a result, one day the entire unencrypted password file came out as the message of the day. The message of the day had been overwritten with the contents of the password file, and vice versa! Very cute little lesson in multi-user programming, which we'll get to again when we get to the difference between Multics and UNIX. So, CTSS was really the inspiration of Corbató. He was a physicist at the time, and became very interested in computers and, of course, wound up being a junior faculty member, then a senior faculty member, and retired just a few years ago. But he's still active. He comes in once a week for lunch with Bob Fano, who's another person from the original Multics era. Fano and Ed David were the ones who wrote the social implications paper of Multics, which was the last of the Multics papers in the Fall Joint Computer Conference 1965 session.

Yost: What about Jack Dennis?

Neumann: Jack Dennis was the key person behind segmentation. All of the stuff that Jack wrote about with Earl Van Horn became the essence of the Multics segmentation machine. Oddly enough, Jack Dennis is still active even though he's retired. He's the thesis professor of Richard Uhler, who's been working summers for SRI on one of our DARPA projects. Jack, with his long-term connection with MIT and segmentation and computer systems, is still very much a presence here. I remember him well from the 1960s, but the fact that he's a thesis professor for a student who is working for us right now I think is a tribute to his longevity. He was certainly there very early on, in terms of the whole concept of segmentation in its true sense. My current colleague, Robert

Watson, has just written a CACM paper on access controls, and capabilities. In fact, I'm co-author with him and others of a new paper revisiting segmentation and making the claim that its time is here again. This brings us back to a capability quote of Butler Lampson, which is that "capabilities are the way of the future -- and they always will be." Indeed, the capability machines of the 1950s and 1960s and 1970s were many years ahead of themselves. But now we're doing it again – somewhat differently -- and we have reason to believe that we can pull it off this time. So, Jack Dennis's legacy is still very much present. In fact, in this new paper that I just referred to, there's a reference to him right up front.

Now we've gotten beyond CTSS and Corbató and Bob Daley, who was very much instrumental and very important in CTSS, and that leaves the transition to Multics. In addition to co-designing the hardware, Glaser really was the person who anticipated much of the software concepts that might exploit the hardware. The software effort didn't begin seriously until January of 1965. But by January 1965 the hardware was fully specified as a retrofit to the GE635, which became the Honeywell 645 when delivered. This was a machine that was visionary in several respects, despite being a retrofit. It wasn't a pure clean-slate architecture, as it just took the existing 635 and added segmentation and paging, and a bunch of other things that were necessary to really support time-sharing in a facile way. You needed things like a read/alter/write instruction where you can guarantee no interference, no interrupts while it was executing, and a few things like that that were embedded into the hardware. And again, Ted Glaser, with his what I am calling infinite wisdom, really anticipated all of the needs of the software. The

ring structure and a lot of other stuff were still pretty fuzzy, as was dynamic linking of symbolic file names. Ted really had a pretty good grasp of what the operating system would have to be like, and what was needed in the hardware in order to support that.

Yost: That's very good to know as far as Multics history has been written, I don't think Glaser is probably properly credited.

Neumann: He's not widely enough known, but he was really an extraordinary influence.

Yost: And he, of course, was a member of the Anderson Committee in the early 1970s, in fact, officially he was chair of that committee.

Neumann: Yes. Ted is really the visionary here in anticipating what was needed. The 360/67 TSS was also a retrofit, onto the 360 series. But IBM never really understood the potentials of segmentation, or the hardware needs for implementing time-sharing; TSS was just a quick and dirty retrofit.

One of the questions I get asked is why didn't Multics succeed commercially? – although it certainly succeeded from a research point of view. The answer seems to be that Honeywell folks had gotten a bunch of large corporations that had always been IBM customers to sign a letter of intent to acquire Multics. IBM salesmen said, we can't let that happen -- so they announced that the 360/67 TSS would do everything that Multics could do, which wasn't quite true. Many would-be Multics customers went back to IBM, to stick with Big Blue as they had done before. So, part of the problem was marketing,

and part of the problem was that Bell Labs had promised that Multics would replace all computing in the labs in something like three or four years, but it wasn't quite ready -- so they canceled the contract and bailed out of the Multics development effort, while MIT and Honeywell kept on going. I think from an intellectual point of view, Multics was a phenomenal success. From a commercial point of view, it was not a great success. On the other hand, Multics in 1965 solved the Y2K problem, thirty-five years ahead. It addressed all sorts of things that nobody else had addressed. The Canadian defense establishment had a five-processor Multics running past 2000, and that may have been the last system extant. But its legacy is still quite visible. If you talk to Tom Van Vleck . . .

Yost: Yes, I've interviewed him, very helpful.

Neumann: Okay. Tom is really one of the important still-active Multicians.

Yost: He's done a wonderful job with the website, the Multicians site that documents aspects of Multics history.

Neumann: Phenomenal job with the website. And he's just another sweetheart, terrific person. I presume he gave you some good insights into Multics. So my experience with Multics was that it was a remarkable collaborative effort. Bell Labs had put a contract to Digitek to create the world's first PL/1 compiler. And at the end of the six-month contract — for which they said would be easy, they'd done lots of FORTRAN compilers — at the end of the six months there was a full page ad in *Datamation*, one of the trade magazines

in those days, and it said, Here and Now, the World's First PL/1 Compiler. Well, their PR people had forgotten to check with their technical guys. The ad guys had been told that the compiler would be completed in six months, but it turned out they couldn't deliver it. They never completed it, and the entire company went belly up. As a result, Bob Morris and Doug McIlroy decided that they could write a compiler for a subset of PL/1 in a few months. We didn't need the whole language -- just what would be needed for Multics. So, they threw out pointers and other features that were overly complex -- again in the sense of Einstein — much simpler, but no simpler than it had to be. The PL subset called EPL, Early PL, really made the Multics programming effort cost-effective. The other thing that made a huge difference was this notion of the so-called triumvirate, which consisted of Corbató and Charlie Clingen and me. Everybody who wanted to write a line of code or who was asked to write a line of code had to submit a detailed written English language specification that identified the purpose of the module, the inputs to the module, the constraints on the inputs, the preconditions, the postconditions, the state changes that were to be taken -- albeit in an informal way. No one would be able to write a line of code until they had approval. Early on, there had been one exception, and that was the dynamic linker. The original version had been written in assembly code, supposedly to optimize its performance. Several major changes to the linkage section were required in the course of the early Multics development. At that point, the person who had written the original linker had left. As a result, the linker was rewritten in EPL by someone else, based on the updated documentation, Much to the surprise of a few remaining assembly language advocates, the EPL code smartly outran the supposedly hand-optimized assembly code. This is interesting, because in a session at the Spring Joint Computer

Conference of 1967, we talked about using the PL/1 subset as a high-level language for developing an operating system. Someone from Burroughs got up (Creech?) and said hey, they'd been doing that for years. And the rebuttal was yes, but Burroughs never published anything about it, whereas the Multics work was public.

As an aside, that conference was where Fred Brooks came up to me and said, Peter, you're an old information theoretician, and you should understand that on information theoretic grounds alone, Multics is impossible! (He had heard that we could interrupt on any character, or on every character – although that was clearly overkill.) I guess, he didn't understand that there was a co-processor (the Generalized I-O Controller, or GIOC) that could interrupt the main processor on each carriage return for teletype command line inputs – which was not seriously disruptive.

Yost: So that was the same conference that Willis Ware and Peters gave a talk?

Neumann: Yes.

Yost: Did you hear that talk? And did it have an influence on you?

Neumann: To some extent, yes.

Yost: Had you thought much about the security problem within the intelligence community?

Neumann: No. Really, that came up later when we got to the Multics retrofit for multi-level security, which involved Jerry Saltzer, Michael Schroeder, Rich Feiertag, Roger Schell, Paul Karger, and a little bit of me kibitzing from SRI. We did a lot of research over the years in multilevel security, as you know, but it was really when the Multics retrofit came up that a lot of those issues surfaced much more visibly. But that was in the early 1970s.

Yost: When the Anderson Committee was working on it.

Neumann: Right. The ability to retrofit Multics to install multilevel security into the commercial offering was remarkably beautiful, in the sense that the structure on the ring mechanism was such that the amount of effort was relatively simple. You had to move a few things from ring one to ring zero, and a couple of things could go from ring zero to ring one, actually increasing the security of the system. I may get back to that issue if we get to multilevel secure database management, but that didn't happen until the 1980s.

Yost: Was Elliot Organick's representation on Multics design, and especially the security design, faithful?

Neumann: Pretty much. Hand me the book; it's right on top of that stack. [points] I would say yes. There are things that he missed a little bit, he misspelled my name, for example, [laughs] but that's not important. But basically, I think he got most of it right.

He knew the players and he talked to us all, and I was very impressed with the way he was able to abstract a lot of it. Maurice Wilkes was another person who hung around MIT – mostly in the summers. I don't know if you have run into him in your interviews; probably a good bit.

Yost: Yes, I haven't interviewed him but . . .

Neumann: Oh, you couldn't. He died a few years ago.

Yost: . . . I was editor in chief of IEEE *Annals of the History of Computing* and he was associated with that journal for a long time.

Neumann: Right. Elliott was really sort of on the fringe of everything but he really absorbed a tremendous amount. I guess he loved writing books. He wrote, what, 19 books or something? That's an outrageous number.

Yost: So you had a short reverse sabbatical and you went to Stanford for a bit?

Neumann: Yes, in 1964. Dave Huffman came out to Stanford to teach for the 1963-64 academic year, and he called me up and asked if I would like to come out for a quarter. So I was a visiting lecturer for the spring quarter of 1964, and I taught a course on error correcting, error-limiting, and survivable coding systems. I had lunch with Huffman almost every day. That was a very, very interesting time. Dave met me, my wife, and our

then three-year-old daughter at SFO, and drove us to Stanford in his open convertible at SFO. (His license plate was “FINITE” like finite math, more than continuous math; even though the paper he did on the continuous deformation of semi-rigid surfaces was continuous math.) We were driving down what was then the Bayshore Freeway from the airport to Palo Alto, and there’s this double rainbow. It was the first time I’d ever seen a double rainbow, and it was spectacular. Dave really liked the West Coast, so when he went back to MIT and he received an offer from Santa Cruz he decided to accept it. I’m still in touch with his daughter Elise.

Yost: What did you think of the West Coast at that time?

Neumann: Oh, my time in Palo Alto and Stanford for that quarter was phenomenal. I just loved it. It was great.

Yost: I figured you must, since you came back and have spent the remainder of your career here.

Neumann: Yes. What happened later was that in 1968 my mother was diagnosed with terminal cancer (she died in early 1970), my first wife and I had divorced, and I wound up with our three kids – whom I would subsequently raise on my own for the next 17 years. Lotfi Zadeh came to visit me at Bell Labs, and asked whether I would consider teaching at Berkeley for a year? That turned out to be a wonderful academic year of 1970—71, and the kids absolutely loved California. I taught five courses. Virgil Gligor

was one of my students when he was an undergraduate, and he took my hardware course. I taught an operating system course twice. I taught an error-correcting code course, and helped run a couple of seminars with Butler Lampson, Jim Gray, Peter Deutsch, and others. I wrote a chapter for a book by Frank Kuo and Norm Abramson.

At the end of my Berkeley stint, Jack Goldberg (who became my first lab director here at the Computer Sciences Lab, CSL) asked me to come down for a week's consulting to help scope out the future of SRI's computer science research in systems, reliability, security, and related things.

Yost: Was SRI then focusing some efforts on computer security yet?

Neumann: Only tangentially, although the beginnings of work on formal methods were already underway. Huffman had been a consultant here when he was at Stanford, and then he became a consultant later, as well.

1964 was when I had first met Jack Goldberg, Bernie Elspas, Bill Kautz, Marshall Pease, and a bunch of others who had been working on array computing and all kinds of wonderful visionary efforts. CSL has a long history, beginning with ERMA, which was the magnetic-ink character recognition system for reading bank checks. And there was a lot of research here on government and commercial contracts in the 1960s. Some of the stuff on array computing was 20 or 30 years ahead of itself, dealing with distributed systems and things like that. So in 1969, here's . . .

Yost: Of course, Engelbart was . . .

Neumann: . . . Engelbart was doing wonderful work at SRI, right.

Yost: How did you like teaching versus pure research?

Neumann: Oh, I loved teaching, sure. My teaching load at Berkeley was rather heavy for a visitor. But I loved it. I was on 10 doctoral qualifying committees. I really got a wonderful taste of academia, quite more directly involved than in my quarter at Stanford.

Yost: That's a lot of teaching and committees in a short span.

Neumann: Yes. But then came the question of what to do next, and the fact that the kids loved California so much made it clear that we could just move down to Palo Alto – where the schools were superb and all within walking distance from where we were living. So I started at SRI in September 1971, and have been here now for what? [More than forty-two years by the time I'm proofreading this transcript].

Yost: And that was a bit earlier than the start of the Provably Secure Operating System project, so what did you do at SRI research-wise leading up to that?

Neumann: Yes, PSOS started in 1973. I'd been at SRI for about five days, and Larry Roberts (who was head of IPTO at what was then ARPA) came by and said they needed a project on fault-tolerant computing. I looked at fault-tolerant computing as a

trustworthiness issue, and wrote a bunch of stuff on how you don't want to just look on it as fault tolerance, but rather as a much broader topic. That was when I wrote the paper with T.R.N. Rao. With Karl Levitt, John Wensley, Jack Goldberg, we wrote a lengthy report, where I wrote something on the hierarchicalization of fault tolerance, and how you have different approaches at different layers, and how abstraction is fundamental to dealing with system fault tolerance. The ARPA contract was just wrapping up when Doug Hogan and Hilda Faust came by from the computer research group at NSA. They wanted someone to design new hardware and new software in a clean-slate architecture (although they didn't call it that then) for very secure and trustworthy systems. So, I basically hit the road running from the day I came to SRI, from the fault tolerance work through the PSOS design -- which ran from 1973 to 1980. The PSOS project developed the software methodology, the hierarchical development methodology (HDM) for PSOS, the specification language, SPECIAL, the SPECIfication and Assertion Language that was used for PSOS, but also for Ford Aerospace's Kernelized Secure Operating System (KSOS), and a few other efforts. Rich Feiertag developed the SPECIAL flow analyzer for multilevel security. We used the Boyer-Moore Theorem Prover to take the flows that had been analyzed by Feiertag's tool and apply it to the Ford Aerospace KSOS, which was specified in SPECIAL. I ran the proofs on that. It took three hours to run the flow analyzer and the proofs over the 34 functions of the KSOS kernel. We discovered that almost half of them had security flaws in the specifications, several of which were easily fixed because they were specification errors rather than system errors, but some of them were covert channels and other problems.

The PSOS project actually went on for another three years until 1983, and what came out of that was a lot of the work that led Rob Shostak to develop what later became PVS, the SRI formal verification system. Rob and Richard Schwartz worked on the verification stuff as a part of the NSA project. The continued funding also supported Joseph Goguen and Jose Meseguer for the 1982 paper on non-interference and the 1984 paper on unwinding. So, that was 10 years of NSA work that really had not just PSOS but this other stuff as well. That was a very productive 10 years of my life, leading that project. Shostak and Schwartz went out to form a startup that developed the Paradox query-by-example database management system. At about the same time, Dorothy Denning and I were involved in a summer study for the Navy's National Academy on multilevel security database systems. Marv Schaefer was the head of that study. Dorothy Denning and I were the head of the futures subgroup, and we suggested that what we really needed were anomaly and misuse detection systems of the kind that Jim Anderson had suggested. But the major innovation was the idea of a multilevel secure database management system. We said what we really should do is take a multilevel secure kernel (which Roger Schell's Gemini Corp had been developing, among others), and put on top of that an Oracle database management system that had no notion of multilevel security. We firmly believed we could develop a multilevel secure database management system that way. Many others seemed to think that would not work, but Dorothy joined SRI and we got funding to do that. We commissioned Gemini to produce the verified A1 kernel. In the process, we discovered that Oracle was sharing buffers among all users, which created some giant covert channels. So, we asked Oracle to make it a per-process buffer. They did. Unfortunately, Gemini was unable to deliver the kernel in time, so we used a

compartmented mode workstation -- which completely threw the baby out with the bathwater from the point of view of A1. But it resulted in an overtly multilevel secure database management system, if you trusted the compartmented workstation. That was one little burp along the way that from a research point of view showed, hey, you can actually build this. And this — the notion in the Orange Book litany of balanced assurance, the Red Book, the Gray Book that Virgil helped write, and some of the others — the idea that you could get more trustworthiness out of something by having an underlying kernel. From a research point of view, this was a very interesting result that never made it in the commercial world, because Oracle found a different approach that would provide less assurance but be much simpler.

I discussed the Descartes quote about the best being the enemy of the good in CACM Inside Risks articles. If you're looking for something that's merely good enough, you may be in trouble – because what we've have today is nowhere near good enough. So this concept of the multilevel secure database went on for quite a while, and has been important part of the research that has somewhat fallen by the wayside as being too difficult.

Then there's the paper that Norm Proctor and I wrote in 1992 on covert-channel-free multilevel security, where you put your trustworthiness for multilevel security in servers. Do you remember Rushby and Randell's distributed secure system at Newcastle? That had lots of covert channels, so it was desirable to do a little better than that, and we came up with something where you put your multilevel secure trustworthiness in servers where all the user systems are single level. You could throw in a compartmented workstation, or

something else if you wanted to, but the point is that we showed that there would be no user covert channels in that kind of an architecture. This is a paper that nobody seems to remember from 1992.

Yost: Yes, I ran across it.

Neumann: And it's on my website.

Yost: I remember in my interview with Steve Lipner, he told me that for Paul Karger and himself working at DEC in the 1980s on developing an A1 system, covert channels were the huge challenge. And ultimately, with financial concerns DEC —

Neumann: Then DEC's management killed the project.

Yost: The project took longer because of a covert channel problem.

Neumann: Yes, covert channels are a bear. So the funny thing is, is very few people have ever referred to that 1992 paper. I should Google citations of it, and see it I can find anybody who's referred to it. But it was, I think, a seminal paper and it sort of preceded the notion of the MILS stuff, Multiple Independent Levels of Security, which Rushby and Rance DeLong have been working on here for some years, along with others who are seeking something practical that does not require the full complexity of true multilevel security.

Yost: To move back a little bit, what were the greatest challenges with PSOS?

Neumann: Well, the greatest challenge was that we knew we couldn't build the hardware, so we just designed it and tried to prove things about it. And at that point, the proof tools were terribly limited. We were not ready to do anything there, but we showed it was very valuable to be able to prove properties about the specification, without ever implementing anything. When we get to what we're doing today, you'll see that that is a very important thing. The good thing was that we learned that we could specify the hardware and the software in a single language, but it was a nonprocedural language. What we can do today is with an executable language, where you can compile basically into hardware or software. (See Nirav Dave's thesis, noted below.)

The real lessons of PSOS were, first of all, that you can deal with a very complex system hierarchically in several ways that are important. One is that in PSOS, every module was formally specified in at most two pages of formal specifications, including all the inputs and outputs, preconditions, post conditions, exception conditions, mappings from one layer to another in terms of the state spaces, and even abstract programs for how to execute each layer in terms of the lower levels. The second thing was that in a capability architecture, you could execute a higher layer as a single hardware instruction -- if the segment tables, page tables, caches, and so on were already set up. So you don't need layers of interpretation in a capability-based architecture. That was one of the really good things.

The bad thing was we were building a single system and we weren't worried about networking. The capabilities and their unique identifiers were unique for the lifetime of the system. That was unreal for 1973, but yet it resulted in a very clean architecture where we could solve what we couldn't otherwise deal with. I think the good thing there was that we convinced ourselves that you could build a capability architecture that was very efficient, that was completely formally analyzed from the bottom all the way up. You could prove properties one layer at a time, and use those properties to prove properties about the next layer, and then iterate all the way up. The fact that it was nonexecutable meant that you then had to write the code independently. When we get to what we're doing today, I'll talk about Nirav Dave's Ph.D. thesis at MIT, in which he developed a single executable language as an extension to the Bluespec language, where you can either compile specifications into Verilog (for FPGAs) or you can compile them directly into a programming language implementation. And you can move the boundary around, experimenting with different hardware-software boundaries. This is really beautiful stuff.

[BREAK IN INTERVIEW]

Yost: You were talking a bit about PSOS and learning . . .

Neumann: The lessons in it. Yes. I think the effort was a very exciting, because we really solved a bunch of problems that had not been dealt with properly before. If you think about the Multics ring structure, you remember that ring one could never clobber ring

zero, ring two could never clobber ring one, and so there was a natural order to the system. Well, in PSOS, we wound up with something like 16 or 17 layers that we specified with that property, sort of like the Biba property where you never depend on something that is less trustworthy. The PSOS hierarchy was a beautiful working out of how you can do that in a really large, complex system. And the essence of it is that there's a sort of a classical paradox where in order to have processes, you need to have memory. In order to have memory, you need processes. And if you're not careful you wind up with circular dependencies. In Dijkstra's 1968 paper, the THE (*Technische Hogeschool Eindhoven)* system had the property that the locking mechanisms were partitioned in such a way that it never depended on something at a higher layer and consequently there were no deadlocks between layers. I remember years later talking with Nico Habermann, who actually wrote a lot of that code. I said, how did that work out? Did it really have no deadlocks? He said, you know, that's funny you should ask. There were actually some deadlocks *within particular layers* that emerged over the years, but there was indeed never a deadlock *between layers*. Clean hierarchicalization was one of the problems that we really dealt with in PSOS. For example, we had a user-process layer and a system-process layer, and the reason for that was you had a virtual memory in between, with segmentation, and you needed to be able to specify in such a way that you can prove rigorously. You never depended on anything that was at a higher layer. One of the clarifying things in PSOS was really resolving that kind of issue. It seems obvious today, but it wasn't obvious in 1973.

Yost: So often true with things.

Neumann: Yes. And we actually had Bob Fabry as a consultant for the very beginning of the project. At the time, he was a professor at Berkeley who had been working on the Berkeley computer capability architecture. Bob was helpful in talking that dependence issue through and helping us come up with a really clean architecture. We showed that you could specify the entire system in a common language. We showed that you could use the Robinson-Levitt paper to demonstrate hierarchical closure, basically proving one level at a time and being able to prove all the way up to the highest layer that you wanted to deal with. That was a very elegant theoretical result and, one that most people today don't know exists. Occasionally I review a paper for the CACM or somewhere else where the authors never knew that that paper existed. On the whole, I would say that we learned a great deal from PSOS. And the beauty of it is that we're using everything that we learned then in our current SRI-U.Cambridge DARPA CRASH project. So I presume we'll get to that at some point.

Yost: I'll be sure and ask about that later. You also were involved in a project with Karl Levitt and Larry Robinson that was funded by the National Bureau of Standard (now NIST) on the hierarchical development methodology.

Neumann: Yes, that was still a continuation of the stuff we did with NSA. At the time, NBS wanted us to write the documentation that we never really got to under the NSA contract, to really formalize the Hierarchical Development Methodology. That turned into a three- or four-volume effort, and it was very good to have that bit of funding because it

meant that we had to put our money where our mouth was and demonstrate how this stuff could be useful. I don't think much of that work has survived, in the sense that it never gets referred to, but it was seminal in the sense that it really characterized the hierarchical architectures and how you can formally specify them.

Yost: In 1980, James Anderson wrote a paper on intrusion detection and then SRI, by mid-decade, was heavily involved in that area.

Neumann: Right. Again, Dorothy and I wrote the part of the Marv Schaefer 1982 summer study where we characterized the need for that technology. Then the CIA came to us in 1983 and said will you build us an insider misuse detection system? -- which we did. It had a statistically based user profile for each user. It took the CIA a year to sanitize the live system data, because it was based on classified systems that they wanted to monitor. They started with IBM mainframe SMF records, where the security officer had a new fanfold stack two or three inches high every morning on his desk, and he had to look through it for anomalies. On the basis of the sanitized data, we built a system for them that would look for anomalous use by any of their agents. Now, it was just about the time we were to deliver that to them that they said, oh by the way, we decided we don't have any insiders misusing computers in the CIA – around the time of Aldrich Ames?

Yost: Would you characterize the CIA project as an expert system?

Neumann: Not in the classical sense. It was statistically based anomaly detection system. However, the expert system came in the next system we built, called IDES: Intrusion Detection Expert System. For IDES, we borrowed an NSA expert system, MIDAS, and re-implemented it. We also hired Alan Whitehurst, the NSA programmer who had developed it.

After that, we built NIDES, the Next Generation Intrusion Detection Expert System, NIDES brought together the intrusion-detection expert system and the anomaly-detection statistical stuff. Ironically, we were subsequently asked by the FBI to put a version of NIDES into the FBI to detect possible insider misuse. So I had done a study about the feasibility of putting NIDES into the FBI Field Office Information Management System, FOIMS, which was a secret-level system (which, it turns out, apparently had top-secret data in it because they had everybody cleared top secret, and they put top secret data in it even though it was secret) [Laughs.] I had shown how to put a one-way pipe from FOIMS to NIDES, and that was the point at which Paul Fitzgerald was the key person in the Bureau who deeply understood computer security. In my opinion, he was the ***only*** person in the Bureau who really deeply understood security. And Louis Freeh informed Paul that because he was also an FBI agent, he could no longer work with computer security – because he was needed as an agent. It was at that point that the FBI decided that they actually didn't have any insiders who were misusing computers, and therefore that they did not need NIDES. This was probably around the time of Robert Hanssen, who is the person they asked to find the mole inside the FBI — which of course turned out to be Hanssen. Then there were two years where Bob Anderson at RAND and I were

involved in running workshops on insider misuse for NSA. Then there were the Dagstuhl workshops on insider misuse, for which I wrote a chapter on the subject for the book. Thus, it's something I have worried about since 1983. And yet I find that almost all of the computer security research ignores insider misuse, denials of service, and survivability issues, which are things that have concerned me since the mid-1950s.

Yost: All along.

Neumann: Yes. So all in all, I would say the history of intrusion detection is problematic. The new stuff that Phil Porras is doing is phenomenal. He's got some absolutely wonderful stuff: the malware threat center, BotHunter, cyber threat analytics, their recent network analysis tools such as SE-Floodlight. Phil came to SRI in 1996, and he took over EMERALD, which was a project we were just getting funded. As soon as we got the EMERALD funding, I hired Phil and he has led our efforts since then; in short, everything to do with network analysis. His master's thesis was with Dick Kemmerer at Santa Barbara. We also hired Ulf Lindqvist from Sweden after he received his Ph.D. at Gothenborg, based on work he done at SRI during two summer internships. So we have a long history; 1983, that's more than 30 years of work in this area.

Yost: In the 1980s, can you put the CIA project and IDES in context with what was going on with intrusion detection work previously?

Neumann: Yes, we've been involved in research for years and years.

Yost: Really, IDES was path breaking.

Neumann: Well, it was really only one evolutionary step. But SRI has more or less been at the forefront of that field. However, my honest feeling is that that work was really trying to solve the wrong problem. If I look at the antivirus stuff — McAfee, and Symantec, and so on — that seems to be the wrong solution to the wrong problem. This takes me back to Multics, of course. We wouldn't be having many of these problems if we had a more meaningfully secure operating system and appropriate underlying hardware that was supporting that operating system. The work that we're doing now for DARPA is trying to leapfrog over all of the present-day vulnerabilities and attacks, and get to something that has sound hardware and secure software, where we might be able to prove that it would be much more trustworthy all the way up.

Yost: I'll be interviewing Teresa Lunt tomorrow . . .

Neumann: Good.

Yost: . . . can you briefly tell me about her contribution?

Neumann: Teresa's role in this was great. Dorothy had been here. Dorothy went off to DEC. Teresa came in before Dorothy left and Teresa became the leader of all the work on intrusion detection until Phil got here. But Teresa had left by then, having gone over to

Xerox PARC. Teresa was a fundamental contributor to CSL. She was also involved in some of our multilevel database management stuff, which she wound up leading. Yes, Teresa was very important. So I presume you'll have a productive interview with her.

Yost: When we took a short break, we discussed very briefly the Oakland Conference Symposium on Security and Privacy. You were there from the beginning. Can you discuss the early formation of that conference, the early development of that conference, what it meant?

Neumann: George Davida and Stanley Ames ran the first meeting in 1980, more or less by the seat of their pants. Rein Turn came in the second year, and helped.

Yost: He was at RAND?

Neumann: Rein was at RAND. I was at SSP the first, second, third, fourth, fifth, and sixth — pretty much most of the early years. I was program chairman in 1982, and Bob Morris was my co-chairman. There was no structure in those days -- it was all get your friends to submit papers,. It was later that Teresa was instrumental in structuring it much more than I had, and it became much more of an IEEE effort with committees and so on. But in 1982 it was still seat of the pants, very informal. That was a great year — for example, the paper by Steve Lipner on how you could use multilevel security and multilevel integrity in a commercial environment, and the Goguen-Meseguer paper on noninterference. There were some absolutely wonderful papers in 1982. And let's see,

Tom Berson came along. He was the one who negotiated all the contracts with the Claremont and kept us there; even after they tried to throw us out, he got us back in. Virgil Gligor was involved from early on. Dick Kemmerer missed the first year, because he was at our first VERkshop (see next paragraph) and could not be away two weeks in a row. The IEEE SSP (euphemistically still called the Oakland Conference even though it is no longer in Oakland and is not a "conference") remains the place to have your paper published if you were involved in serious security research.

At the time, I was running the first two so-called VERkshops (Verification Workshops). Have you run into those yet? The proceedings appeared in the ACM SIGSOFT Software Engineering Notes (for which I was the editor). I ran the first two with Steve Walker, who was in the Pentagon by the second one after creating and running Trusted Information Systems (TIS). Karl Levitt ran the third VERkshop. These three meetings were an attempt to bring together the formal methods people and the systems people — Marv Schaefer, Dick Kemmerer, Bob Boyer, Don Good, and lots more. Do you have all the ACM SIGSOFT Software Engineering Notes?

Yost: No, we don't.

Neumann: I created that in 1976 and edited it for 19 years. I turned it over to Will Tracz who edited it for 19 years and he has now turned it over to Mike Wing. [points] So that's this whole second shelf, from the blue on the left to the half-blue on the right

Yost: Before SSP, obviously there was sharing of papers informally and the National Computer Security Conferences.

Neumann: Those were run by NBS and NSA conferences for many years. They go back a little bit further.

Yost: Back into the mid-1970s?

Neumann: Let me see when was the first one. … Well, I'll have to dig it out and take a look at all that. … Here's the twelfth. There were two a year in the beginning. This one is 1989, so I think they might slightly predate the IEEE SSP, in 1978, 1979, something like that. We'll have to check that later. If I go up there and start pulling stuff down, it all falls off the shelf and gets to be a mess.

But those were very important conferences in the beginning. Now, from the point of view of NBS/NIST, NSA, and the U.S. government, it became a trade show and unfortunately lost its technical element, and then they abandoned it. They are trying to recreate it in the CyberMaryland conference that started two years ago. That's sort of a local attempt to reconstruct the national computer security conference.

Yost: To have it more intellectual rather than trade show.

Neumann: Maybe. But it's still somewhat of a trade show, I think. The one I was at in 2012 was dealing more with short-term problems rather than long-term thinking, whereas the original NCS had a lot of longer-term thinking. But the real long-term stuff is Oakland, or it was. Now that's become somewhat shorter-fuse, too. You see more incremental solutions that had been started in the middle to late 1980s, when you started getting the papers on polyinstantiation for the database stuff. Teresa will talk with you about that; she's probably still upset about that. [Laughs.] You get her to talk about that, and that's fun. I don't want to put words in her mouth. You probe her a little bit she'll probably open up on it.

Yost: Now, of course, in recent years, RSA is also more of a trade show. In the early years, was that was more of a small cryptographers conference?

Neumann: I chaired the cryptographers panel nine out of the first ten years with Diffie, Hellman, Adleman, Shamir, and later Thalia Rabin— wait a minute, she's Rah-BEAN, but her father is Michael RAY-bin. Yes, I'd often twit them a little bit about cryptography not being enough by itself. There's this wonderful quote that Butler Lampson attributed to Roger Needham, and Roger attributed to Butler, "If you believe that cryptography is the answer to your problems, you don't understand cryptography and you don't understand your problems." Beautiful, absolutely beautiful. So, I chaired the panels when they were first talking about crypto algorithms and technology. And now, I think, that's become a trade show. It's just a commercial show with huge vendor offerings in the exhibition hall part of it, and parallel sessions with people talking about

things that the marketing guys don't understand. So I'm afraid that at the cost that they charge, it's not worth it unless you're in the business, and SRI's not really in the business. We're a not-for-profit research organization. So the RSA conference, in the beginning, was really a sort of club and had all the people who were actively working in the field invited as participants. Now, it's really gotten to be much less exciting from a technical point of view. It may be really wonderful from the business point of view, except they're selling stuff that may not be good enough.

Yost: Much of what you've talked about gets at the idea of what you're quoted as saying is a holistic approach to computer security. You mention that there are some others that see things in a similar light but unfortunately, many don't. Who are some of those like-minded individuals and what have you learned from them?

Neumann: Earl Boebert. Do you know him?

Yost: No I don't.

Neumann: You should. Earl's retired, but he then went to work at Los Alamos or somewhere down there (boebert@swcp.com). He was part of the Honeywell crowd that spun off Secure Computing Corporation. Virgil Gligor. Jerry Saltzer. Peter Denning. Lots more, also other people who did have a role long ago but aren't around much anymore. I guess you're looking for other people you might interview. Boebert would be great. In

fact, he just wrote something that I'm putting into my *Software Engineering Notes* risk section, which I do every two months. Are you reading the ACM Risks Forum?

Yost: Yes.

Neumann: Okay. Well, he had an item a couple of issues ago. It's called "How Do You Code a Secure System?" in *Risks*, volume 27, number 25, which you can dig up at risks.org. I'm publishing that in the next issue of *Software Engineering Notes*. Boebert really had a great deal to do with all the early security stuff. They also adopted one of the key features of PSOS – putting strong typing into operating systems and applications -- so that you had an object-oriented system, with strong typing. One of the main things that Honeywell did, and then the Secure Computing Corporation, in the Secure Ada Target and the Sidewinder firewall, was the use of type-based security controls. The notion of strong typing in hardware and software systems seems to have originated with PSOS, particularly embedding it in the hardware. Again, this is something we're doing now in our current DARPA CRASH project. We're going back to that and finding a way of putting it into the hardware in a clean way that is respectful of the needs of programming languages, for example.

Let's see, who else? Corbató, of course; he's retired, but still very, very viable. He will tell you that he doesn't remember very much, but he remembers almost everything. He would be a very good person for you to interview. When I did the 2013 Elliott Organic memorial lectures in March in Utah, I had a lovely time with Steve Corbató, who is

Corby's nephew, and I said why haven't you invited Corby out to do the Organic lectures? He said yes, you know we should have done that.

So let me think -- who else? Lot of younger folks; I'm trying to think of the people with the real intellectual history. I could go back and look at, say, the early Oakland conferences but you've got Teresa, you've got Gligor. He is really outstanding. Carnegie Mellon now, and after years at Maryland. Yes. I can give you a long list but you've probably got most of them already, I would think.

Yost: What about Carl Landwehr?

Neumann: Landwehr, oh Landwehr is great. He might be conflicted, of course, because he funded you.

Yost: He did, but he's no longer running Trustworthy Computing . . .

Neumann: That's right. So you're not —

Yost: . . . We felt we couldn't interview him, unfortunately when he was our program officer, but plan to ask him now that he is out of NSF . . .

Neumann: But now you can. Oh you should; you must interview him. He's written some really good stuff, and he had an important role in the Naval Research Lab.

Yost: Right.

Neumann: There are a lot of specialized guys like David Bell and Len LaPadula. There's some other folks from that era. John McLean would be good at Naval Research Lab. He came up with the Zed System, System Z, which totally demolished some of the Orange Book oversimplicity. He's got some nice views on reality. Let's see, who else? Karl Levitt, who you probably talked to.

Yost: I am this trip.

Neumann: Okay, good. Karl's presence was seminal in our SRI Computer Science Lab.

Yost: Can you talk about him as a colleague and his role in . . . ?

Neumann: Oh, he was tremendous amount of fun to work with. He's a very delightful person. Karl is rather disorganized, but a sweetheart.. He was once something like two years late on a final report -- before the government got to the point where they said if you're late, we'll cut off the entire company. I basically sat down at a keyboard with him and said look, what's missing from this report? He said, I've must write a paragraph. I said, this has taken you two years? He said, yes, I've got too many other things going on. This is when he was the Associate Director under Jack Goldberg and I was the Assistant Director for a while, back in the PSOS years. Karl is very delightful, wonderful, caring. And he was very much involved in the Software Implemented Fault Tolerant (SIFT)

system, which I didn't tell you about. I mentioned Al Hopkins, who at MIT's Draper Lab led the design of the Apollo Guidance Computer. Al built some ultra-reliable systems, which competed with our lab, both funded by NASA. SRI's SIFT took seven Bendix Aviation's avionics processors off the shelf, with seven buses, seven memories, and seven power supplies,. And two-out-of-three voting in all critical tasks. We actually formally proved that, given the probability of failure of the hardware 10 to the minus five, the system as a whole achieved 10 to the minus 10 probability of failure per hour because of the architecture — seven-way replication with two out of three voting. And that was the project that funded Shostak, Pease, and Lamport on the byzantine agreement algorithm work. That was also a very important project. I had a peripheral role in the early days of SIFT. I can't claim deep involvement, but CSL built the prototype and shipped it off to NASA at Langley Air Force Base, where it ran for something like 25 years -- apparently never failing or having to be rebooted. You could disable a processor, memory, or bus, or turn off a power supply; the system was self-diagnosing, self-reconfiguring, and just kept adapting. Karl was instrumental in that.

Yost: I look forward to interviewing him on Thursday. I have an interview with both him and Matt Bishop at Davis.

Neumann: Okay. Matt's another fine guy. When are you seeing Teresa?

Yost: Tomorrow.

Neumann: And what's this afternoon?

Yost: I don't have anything this afternoon.

Neumann: Okay, well we may still be going at this rate. It's 11:30 now, let's see how things go. I have a lot of work to do, so I can't give you the whole day. [Laughs.]

Yost: I realize that and I am very grateful for your time.

Neumann: Well we must get to the work we're doing now, because this is the top of the line excitement for me. After 80 years of being on this planet, I'm suddenly challenged more than I've ever been, and once again doing some really, really exciting stuff.

Yost: Some pioneers we've spoken to on this project, including some members of our advisory committee, have talked about the computer security field as full of factions and there almost being a religious fervor.

Neumann: Yes. Well, I would think you've got Donn Parker. He and Bob Courtney and Bill Whitney and some of the IBM folks for many years were saying that there are no real technical problems -- the real security problems are administrative. Donn once said that there's been no meaningful research in computer security in the past 20 years, in a public meeting. I was sitting there thinking hey, Donn, that is certainly an overstatement. But

that's one faction. Then there was the NSA faction, all of The Orange Book and Rainbow Series devotees. Roger Schell still somehow lives in that world.

Yost: Do you find that the formal methods crowd, most of them really don't take a full holistic view of security?

Neumann: Well, that depends. Our lab does. I mean, Rushby, having worked in Newcastle and Manchester, has a remarkable view of aviation, safety, reliability, security, formal methods — deep formal methods — really remarkable stuff; networking, hardware down to the bits, ions, whatever. He's got tremendous grasp of the issues. Shankar was co-PI with me on the NSF project that we did, ACCURATE, for the electronic voting stuff. Shankar's very much into the real world, despite being a superb theoretition. Bruno Dutertre is the creator of the Yices, the SMT software, and he's worked on all kinds of real problems. Our Lab director Pat Lincoln himself has his fingers in an amazing amount of real-world stuff. So I would say the formal methods guys here don't at all have that problem. I would say that there are a lot of people in security who have this problem, not just formal methodists. I mean, I remember being on an NSF panel where we were wrapping up our cybersecurity proposal evaluations. Somebody came in from the NSF networking group with a proposal that they had evaluated, but wanted a second opinion. We looked at it and more or less unanimously came to the conclusion that this was a fine networking proposal, but that it could never be implemented with any sense of security. Similarly, some of the intrusion detection work would benefit from a broader system perspective. It's not effective in the real world, even

if it is interesting research. Now you have the whole realm of software-defined networking. Much of that work seems to have ignored security until recently. It's a wonderful idea of being able to redefine network configurations on the fly, but doing it in a way that is secure, reliable, and demonstrably sound is a totally different ballgame. You've got some networking people who don't understand security. You've got some security people who don't understand networking. And as I said earlier, you've got some cryptography folks who don't understand networking, or security, or reliability, because they are primarily theoretical.

I would say in answer to your question that many young researchers in the universities are of course interested in getting tenure, and they have to write lots of papers. Some of those papers cannot be applied in the real world, but that's not the role of universities in general. I'm not faulting them, I'm saying that this is a generic problem. Look at the math world. Mathematics departments around the world tend to have very little direct sense of applicability, and many applications turn out to be something that comes out of economics or physics or some other field entirely. The stuff on error-correcting codes, for example, was based on existing mathematics that turned out to be exactly what was needed -- for example Group Theory and Galois fields. Such useful mathematical theory is also fundamental to cryptography. But many mathematicians do research, which they teach to their graduate students, who go off and teach it in their university, with very little sense of what is the potential application of all of this? In short, I love theoretical research that turns out to have superb applicability, but you often never know ahead of time when that might happen. I'm casting a broad brush, here; it's true of a lot of fields, not just computer science. But I would like to see more analysis tools that work in the

real world without monstrous false positives and false negatives, especially when you've got virus writers using existing antivirus tools to check whether they're vulnerable to being caught. They're typically always a step ahead of the defense, which is a generic problem in computer security -- because the attackers always seem to have an edge over the defenders. Defenders have to seal off every possible real vulnerability, whereas attackers just need to find a few weak links. Right now, one of my favorite statements is that DoD is always talking about defense in depth and defense in breadth. On the other hand what we have is *weakness in depth and weakness in breadth*. That is actually a segue into the stuff we're doing now where our DARPA Program Manager, Howie Shrobe, who started the clean-slate programs, came to the conclusion that you can't get there from here. If you want trustworthy systems, you can't do it incrementally by patching and trying to make the silk purse out of the sow's ear, as it were. Okay, that's sort of the long answer to a short question, for which I'm known.

Yost: I've just got a couple more questions, and then I'll ask about current projects. The first is about your ACM Risks Forum. It must be one of the longest running newsgoup, if not ***the*** longest running one.

Neumann: It could be.

Yost: I'd like to get some context of how that got started, what it's meant to you, and what you see as its role and its impact.

Neumann: For the ACM Risks Forum, it all started in 1985. Adele Goldberg came to visit when she was president of ACM. The ACM Committee on Public Policy had existed for some time. Daniel McCracken was the denizen of that committee in 1983-84, with Tony Ralston and others as predecessors. The ACM Council had decided they wanted some sort of visible online presence relating to public policy and risks. I said great, I'd be happy to do that, so they appointed me chair of that committee, which I've been since 1985. I started the Risks Forum on August 1st, 1985, and the very first issue had contributions involving Dave Parnas, Peter Denning, and Jim Horning, all three of whom became part of my committee. And they still are except for Jim, who passed away recently. Nancy Leveson, Jerry Saltzer, and Lauren Weinstein have also been with me since the beginning. Over the years, it's lost a few people. Kevin Fu has just joined us. Basically, it is a group of folks with fairly strong views about everything, and they have been the editorial board for all Inside Risk columns in CACM. Occasionally, I'll send them an item that has been submitted to RISKS, asking should I run this or is this going to cause too much blowback, or is it simply not appropriate? I get wonderful feedback from them, and they get into marvelous discussions among themselves as well. Every time I get a new column in the works, I get some very lively feedback. They've all been very good at that. Jim Horning was fabulous. So this committee has served as a sounding board for the Inside Risk columns and to some extent, the online RISKS. The very first RISKS issue declared a wide range of things in scope. In those days, missile defense was important -- the Strategic Defense Initiative [SDI]. Electronic voting showed up immediately in the first issue. As I said, this is one of the things I think is really going to be a nasty problem. In 1984, David Burnham had written at least five articles in the *New*

York Times on election fraud, which got me going. I've bird-dogged the need for greater election integrity for 30 years now. Hospitals and medical care, finance, safety and aviation -- all of those were issues that I raised explicitly in the first RISKS issue in 1985. That followed onto the ACM SIGSOFT Engineering Notes, which I started in July of 1976 -- which always had horror stories of things that failed, including items from the 1970s and 1980s. So it was a very easy transition from the Software Engineering Notes documentation of historical events to the online Risks Forum. We included things such as Jack Garman's beautiful six-page piece on the failure first shuttle launch, where they couldn't synchronize the backup computer with the four primaries. It turned out to be a one-in-sixty-four probability of failure that was known to the designers, but never told to the operational crew. All they had to do was reboot it, and it would've worked. Instead, it took two days to diagnose it, and then re-launch. We've also discussed the ARPANET collapse of the 1980s, details about why that failed — a combination of a bad garbage collection algorithm, two bits dropped in sequence numbers in memory, and no memory parity checks. This was a case where the entire ARPANET was down for four hours as a result of something the developers had said could never happen. And then there were all of the power failures from 1965 on. In 1965, I was standing in front of the computers on the top floor of MIT's Project MAC at 545 Tech Square, when the Northeast blackout hit. All of the disk arms started collapsing. Since then, we've had what? Ten, maybe twelve major outages affecting either all of New England or even the entire West Coast, Baja California, and Western Canada — major outages. Each time the providers seem to have said, well, we've discovered what went wrong; we've fixed it, and it will never happen again. I've been documenting stuff like this since 1976, and if you look at my book

(Computer-Related Risks), which was published in 1995 — it'll be 20 years old soon — a lot of the problems noted in that book are still happening. We keep seeing the same kinds of mistakes made over and over again. For example, buffer overflows are still common. (I must note that buffer overflows were prevented in Multics in 1965: there were no buffer overflows in the stack because of the stack discipline -- if it wasn't in the stack frame, it was not executable. Also, we knew back then that we would have a Y2K problem in 2000, and we solved it in Multics in 1965. In a lot of other places, it didn't get addressed until the waning weeks. In some cases, it still caused trouble after 1 Jan 2000, where things hadn't been properly remediated. So the popular view was that nothing happened, it wasn't bad, it was a non-problem; trying to fix it was a waste of money. Actually, it would have been a huge problem if it hadn't been remediated as thoroughly as it was. The irony there was that the Department of Defense had the worst record in its preparations, according to the GAO report cards. They didn't even know how many computers they had. They had no idea how many computers were subject to failing on Y2K.

There's a long history of things that have gone wrong, and I would say a colossal lack of awareness of what might be done to prevent them in the future. We tend to be very short sighted. There's very little effort to do long-term research, and that's become increasingly a problem over time. It used to be that you could do long-term research. SRI, in the array computing stuff in the 1960s, for example, was way into the future, looking at systems that hadn't ever been conceived by others at that point. And yet, we are in this bind where computer security has been ignored in the real world for years. We have would-be solutions that don't solve the problem, and we have all this antivirus stuff that doesn't

avoid new or mutating viruses, and we have man-in-the-middle attacks because people tend to ignore them. Also, we have many denial-of-service attacks because those are too complicated to worry about. And we've got massive insider misuse problems that are pretty much ignored.

Yost: Way back in 1970 in Willis Ware's Defense Science Board Report, toward the end of the report, he expressed a need to keep research open and industry needed to be part of the solution. And you look at what's happened in the industry, that project at DEC was terminated for financial reasons [in the 1980s] and IBM, out of the 1974 SHARE meeting, develops or creates a product with RACF which most people feel in its early stages, and maybe even later, is not very secure. Is there any way to better align what can be done in industry with what is needed?

Neumann: There's a natural dichotomy here. Some people believe that free enterprise will solve all problems; other folks say you can't get there from here starting with what we have now. There's also the tension between hardware and software. If you go back to the Intel 432, we had some hardware architectures with which you could implement a Multics-like ring mechanism, where you'd have more than a user state and a supervisor state, where you could in fact create layers of protection in the hardware. That stuff all went by the wayside. What we're doing today in our DARPA project is taking a MIPS RISC architecture, and adding capability instructions and registers to it. All of a sudden, it looks as if we can rescue the RISC architecture and find that it's actually very appropriate as a starting point for our new hardware. The tension between hardware and

software is that if you think you can solve your problem in software but you don't have hardware that enables that, you're still missing a key link. I think that the fact that we're able to design and implement new hardware, at least in FPGAs, is one of the huge advances. We write our hardware specifications in a higher-level language (Bluespec) and compile them into Verilog, which gets plunked in the FPGA, and if you don't like it, you go back and redo it. And you keep iterating on this until you have something that works. In our CRASH project, we are putting formal methods into the development chain; we have the ability to analyze on the fly the validity of our hardware designs, whether they satisfy various hardware properties that we need in order to have the software properties that must be assured. I think this is a huge advance, and we are pushing at that very hard at the moment. But the idea that —

Yost: When did this project start?

Neumann: Our CRASH (Clean Slate Resilient Adaptive Secure Hosts) project started in October 2010; our MRC (Mission-oriented Resilient Clouds) project started in October 2011. The first one is the one in which we're building new hardware and operating systems with a hypervisor/separation kernel in the style of John Rushby, where you can actually formally prove properties about the hardware and the software. It goes back in part to the PSOS concept of a tagged, typed capability architecture, which is actually overlaid onto a MIPS RISC architecture. We now have hardware specifications written in Bluespec for the capability architecture, and the ability to prove properties about whether the hardware satisfies the properties that we need. So this is something that would not

have been possible when we designed PSOS – which was much too far ahead of its time. And yet today, we think we can solve some of these problems. The problem, I think, is that if you just rely on existing hardware, you can build things that are pretty secure, if you assume that you don't have any insiders, and if you don't have any denial-of-service attacks, and no man-in-the-middle attacks. So you start to get into something where you need more than you have. The challenge that we're confronting with this clean-slate notion is if you were to start all over again, what would you do differently? We're taking that quite seriously in going back and saying, we've learned a great deal in the past 40 or 50 years of hardware and software development. Let's use everything that we know and put it together into a new way of developing hardware and software. I think Nirav Dave's 2011 MIT Ph.D. thesis (A Unified Model for Hardware/Software Codesign) [that's co-design, not code-sign] is the really fascinating part of this, where you have an executable language extending Bluespec, in which you can specify the hardware and the software, and you can compile the hardware into Verilog, which can be put into an FPGA, and you can compile the software into a real programming language and compile it executably. That's something PSOS could not do. Nirav's done it, and one of the subsequent MIT PhD students has built a better implementation. But we're not using it yet to the extent that we could, and I think that's one of the challenges we're going to be looking at in the remaining years of our joint SRI-U.Cambridge projects.

Yost: And what do you see as the impact the project can have and on what different communities?

Neumann: This is a very interesting question. Robert Watson at Cambridge is the real driving force behind our CRASH project, and he's leading all the folks in England who are working on it. He and some of his colleagues have built Capsicum, which is a capability-based operating system that runs on commercial hardware. Now, the realization is that, while it's not secure enough, because you can't trust the hardware, it does provide a sort of an existence proof that you can build a very fine-grained capability system, in which you can implement least privilege in a way that really protects against many of the common characteristic vulnerabilities. And if you go back to Paul Karger's IEEE SSP 1987 paper, he talked about using access controls on procedures. If you are going to take a piece of software (which could potentially be malware) into a container of some sort, you want to be able to know that it can access just those resources that it should need if it's going to do what it's supposed to do -- and nothing else. For example, it might not need to have access to the Internet, or send e-mail, or read your e-mail contacts, or whatever. So this is the notion of least privilege on procedures; what you can do with a capability architecture is you can put least privilege on everything, and you can confine things so that a piece of malware is confined — it's in a contained environment or a sandbox, whatever you want to call it — where the only thing it can do is what something in that container is supposed to do because it doesn't have access to anything else. Now, if you believe in antivirus protection, but you don't know what kind of a virus you're going to protect against, you've got a fundamental problem because your operating system can't protect itself against itself, and much less against malware that's inserted by somebody who clicks on something that he has no idea what it is – and that now has access to all your default privileges. If you can sandbox it in such a way that it

can't do anything harmful, you would be much better off. So one of the things we're trying to do is to build systems where that kind of architecture is easy to implement, where you can specify in detail what it is that the security policy is supposed to be for a particular piece of code rather than just for a user, or a networking connection, or whatever.

Yost: In terms of the economics of the applicability of this, is it something that can be implemented and used economically? —

Neumann: Well, it's interesting. We've just written a paper— I mentioned it earlier when you were talking about Jack Dennis — called The CHERI Capability Model: Revisiting RISC in an age of risk — and it's a justification of how we have gone back and revisited the Dennis-VanHorn type of segmentation and capability architectures, and how this could allow us to implement things that we otherwise can't implement. Segmentation done properly is what you might say is the purpose of that. I think going back in history to things that were too far ahead of themselves, years ahead of themselves at the time but theoretically very sound, if we can apply some of those things to real developments today, we may be able to come up with things that are really orders of magnitude more trustworthy than what's out there at the moment. And again, Howie Shrobe's notion at DARPA that you can't get there from here by minor incremental changes suggests that clean-slate architectures are in fact what is necessary here. And so we're doing that to the extent that we can. Also, ours is a hybrid architecture that enables legacy software to coexist safely with very trustworthy applications.

Yost: That's terrific. Are there any other topics I haven't brought up or any topics that we discussed that you might —

Neumann: Oh, yes, there are tons of other topics like electronic voting, health care, drones, and lots more.

Yost: Many things that have appeared in your Risks Forum.

Neumann: Yes. All of the issues of security, reliability, safety, interoperability; demonstrable composability is one of the most fundamental things here. I've got a report on my website, 2004, when Doug Maughan was at DARPA. He had a program called CHATS, which was Composable High Assurance Trustworthy Systems, and I did a study on what does it take to have some notion of predictable composability. Can you plunk together two pieces and have some derivable properties of what is going to happen? The problem is that human safety and reliability and survivability are emerging properties of the system as a whole, and of the way in which it's being used, and all of the people who are using it, and all of the would-be attackers who are trying to bring it down, as well as environmental problems. You can prove all sorts of local properties of pieces of the system until you're blue in the face, but when you put it all together in an operational mode, all bets are off, basically. One of the real challenges in composition is having predictable behavior when you put things together. Ideally, we're getting to the point where programming languages seem to become less important in the larger scope of

things, where people want to plunk together a bunch of stuff that they can pick up off the Web, and build a system without having to do a lot of programming. Now if you can't have components that, because of the way they are designed and implemented, you believe will be predictably composable, all bets are really off. For example, if you take two supposedly very secure subsystems and you put them together with an unencrypted wire, you've got man-in-the-middle attacks and perhaps other problems. So you put encryption in the middle and now you've got denials of service, and you're forgetting the fact that there are insider misuse problems on both of the subsystems. So I think one of the biggest problems here is this notion of how do you have any hope of composing systems out of components in some predictable way where you don't essentially throw the baby out with the bathwater when you just grab stuff off the Web. Of course, you still have the problem that you just can't trust it. It may be already compromised and filled with malware. I think the answer to that last question is just about every application I can think of, where people will say, well it's not that important, why do we have to worry about it? Medical health is clearly a critical issue. Electronic voting is evidently a potential disaster, in which potentially every step in the process is a potential weak link. People talk about wanting Internet voting, for example. Here you can't trust the machines. You can't trust the ballot you've been given. You can't trust the people who are handling your ballot. You can't trust the environment in which you're operating. You can't trust anything on your laptop. You've got somebody buying your vote and they want proof that you're voting as they've directed, so they're standing behind you while you're casting your vote. Coercion, vote selling, vote buying, all of this stuff is problematic.

The idea that we're going to come up with simple solutions to very complex problems is fatuous. I think one side comment here would be that simplicity gets complex quickly. As soon as you believe you have a simple solution to something, you're probably oversimplifying. So we go all the way back to the Einstein quote that we started out with earlier; it's the **'not simpler'** part that bites us very often when we're saying that we really don't need that little exception condition because it'll never happen. For the exception condition on the original shuttle launch, the one in 64 probability, if they had put in a detector that checked for that mismatch and triggered an automatic reboot, it would've seamlessly solved the problem. The fact that the designers knew about the fault mode but never told the operational guys, because it's only a probability of one in 64, how could something go wrong? Okay, well, in our world here, anything that can go wrong is likely to go wrong. And we've got the Heisenberg problem as well, where you try to test it and testing it shows that there's no problem, but because the test changed the state of the system, and that disaster recurs only in a different context. So, I'm having fun doing all this stuff and I'm going nuts trying to solve all of the problems at the same time. But I realize that if you don't try to solve more than the narrow little problem that is typically the subject of research, your system is likely to be able to collapse of its own without even being attacked, or it's going to be attackable. Anything else?

Yost: Thank you, Peter, this has been very enlightening. I really appreciate you spending the time.

Neumann: Great. I really enjoyed this process. I would comment that despite the desire that nothing should be simpler than it needs to be, we have nevertheless oversimplified considerably in this discussion. I hope you will keep digging for more details!

[PGN, Editing completed 6 December 2013]