An Interview with

THOMAS VAN VLECK

OH 408

Conducted by Jeffrey R. Yost

on

24 October 2012

Computer Security History Project

Ocean City, New Jersey

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

Thomas Van Vleck Interview

24 October 2012

Oral History 408

Abstract

Thomas Van Vleck is a time-sharing and computer security pioneer. As a user he worked with MIT's Compatible Time-Sharing System (CTSS) and MULTICS as a MIT student prior to helping to design enhancements (including security enhancements) to the MULTICS system first as a technical staff member at MIT and later on Honeywell-MULTICS as a technical staff member and manager at Honeywell. The interview discusses the security issues/risks on CTSS that resulted in modest changes (password protection) to CTSS and influenced the far more extensive security design elements of MULTICS. His long association w/ MULTICS in both the MIT and Honeywell setting provides unique perspective on the evolution of MULTICS security over the long term. He also briefly discusses his post-Honeywell career working on computer security as a manager at several other firms.

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. This interview is being sponsored by the National Science Foundation as part of the Charles Babbage Institute's project, "Building an Infrastructure for Computer Security History." I'm here this morning, October 24, 2012, with Tom Van Vleck at his home in Ocean City, New Jersey. Tom, I see that you went to Hinsdale Township High School. Are you originally from the Chicago area?

Van Vleck: That's right. I was born and brought up in Hinsdale, Illinois.

Yost: In school; in primary, middle and in high school, did you have a special interest or affinity for mathematics?

Van Vleck: Yes. I always thought it was beautiful and interesting.

Yost: Would you characterize that as one of your favorite subjects in school?

Van Vleck: I think so, yes.

Yost: When you started your undergraduate studies at M.I.T. in 1961 did you know from the start that you wanted to major in mathematics?

Van Vleck: I did. I had no idea what mathematicians actually did. I'd never met one. And when I found out, I found out how smart you had to be to be a mathematician, I realized that I probably wasn’t in the top tier. Now, when I was in high school back in Hinsdale, there was a local engineer who had formed an informal group of kids that met in his basement and they had started out as a ham radio club because he was a radio ham; and then they got interested in computers. This was back in the 1950s. The engineer’s name was David M. Boyd, and he was the chief instrumentation engineer for a company called Universal Oil Products. Mr. Boyd taught the kids about what computing was about, and they built a simple relay computer probably about 1957. Then they got interested in doing something more and more interesting. Mr. Boyd had lots of industry contacts and the group and he formed the idea of building a transistorized computer. This was a big deal back in 1957, 1958. He had some friends at Texas Instruments who gave him — donated — boxes and boxes of production line dropouts; transistors that weren’t good enough for portable radios but they were great for switching because if you just ran the transistor on and off, it was perfect. And the Bryant Chucking Grinder Corporation donated a Bryant drum, which was a huge drum. It had perhaps 4,000 bytes of memory on it and we were going to build a computer using that drum, basically, as its memory and using the transistors to build logic circuits. I was brought into the group, not right at the beginning — I didn’t get in on the relay computer — but I joined the group and the first thing they did was they handed you a flip flop. A flip flop in those days was a 3X5 or so circuit card with two transistors and a bunch of other components on it. Then they handed me a blank circuit board, and showed me where the components were, and said make one just like that. And they taught me to solder. In those days if you wanted a computer of your own, you had to solder, that was all. So then after I had made my first flip flop, they showed me how to set up a Tektronix oscilloscope to see whether or not it worked. And boy, the first time I got that square wave coming out on the oscilloscope I was hooked. [http://www.multicians.org/thvv/boyd.html] So we worked on the transistor computer and then, you know, as time went on, various members of the computer club moved on, went off to college, and went on various interesting careers. And I went to M.I.T. I knew I wanted to do computers. I visited M.I.T.'s computer center, which was a gigantic room full of one computer, a 709, and (pause)

Yost: Was that influential in your decision to go to M.I.T.?

Van Vleck: I think so. I think that when I did the college tour, certainly the M.I.T. tour guide walked everyone past the computer room and here was this big machine behind the glass, you know the kind of thing they used to do. And that was for me. The IBM 709 was on the first floor of M.I.T. Building 26, and on the second floor was a room that had the TX-0 computer and a PDP-1, which had been donated by Digital Equipment. And there was a group of hackers, many of whom had trained at the Tech Model Railroad Club, and then had gotten interested in switching and signals and had drifted off into computers. But it was well known that a lot of those guys were flunking out because it was so addicting. And so I could have gone in that direction and I just kind of held back until I figured out how to be a college student and probably that's why I didn't flunk out. But I took every computer course that M.I.T. had. In those days there was no undergraduate degree in Computer Science.

Yost: What was your first computer course that M.I.T. had to offer?

Van Vleck: The first computer course was called 6.41. All M.I.T. courses had numbers and 6.41 was "Introduction to Automatic Computation"; and I took that the first chance I could take it. And they taught us FORTRAN-II on the IBM 709. The exercises were fairly simple but they started us out with the Rochester machine, which is an abstract machine, has 100 locations and used decimal; and you learned computer architecture from machine language up.

Yost: Do you recall what faculty member taught this course?

Van Vleck: I think that Professor John McCarthy was listed as the faculty member, but in fact, all the lectures were given by a — I'm not sure whether he was a graduate student or what — named Goldstein, and I don't think I ever saw McCarthy.

Yost: In your first year did you have any direct interaction with M.I.T. computation lab and Fernando Corbató?

Van Vleck: No, I didn't.

Yost: When did that start?

Van Vleck: Well, the way that one made a computer career in those days was that you got a part time job programming, and I got a series of part time jobs programming. First at Project MAC, which would've been in the fall of 1963; and there I worked in a group run by Professor Herbert Teager. Teager had a million different ideas and didn't direct his people to do anything in particular, so I hung out with the guys and did a lot of stuff, but he wanted to build his own operating system — and that never went anywhere. So the next year I got a job programming for the M.I.T. Political Science Department. And I had summer jobs, also, in the computer industry; every summer. First at Universal Oil Products, for a couple of years; and then after two years there I got a summer job at Swift & Company Operations Research Department. These were all back in Illinois.

Yost: Can you discuss what type of programming you did for these companies?

Van Vleck: Well let's see; at Universal Oil Products I started out — having taken two computer courses I wasn't exactly a fully-blooded programmer — so they started me out as an operator and programmer, and I took to it very fast. Universal Oil Products had an IBM 7070, which is a little-known machine. [http://www.multicians.org/thvv/7070.html] The 7070 was basically an IBM 650 re-rendered in transistors and with a core memory, which made it blindingly fast compared to the 650, which was a drum machine. The IBM 7070 was a sweet machine; but decimal, business-oriented, it had asynchronous I/O so that you could actually start a tape rolling and the card reader running; run them both at the same time, which was considered very fancy for the day. And the other thing that Universal Oil had was, it used an IBM 1401 as a peripheral and kind of tape machine; and also for a lot of replacement of traditional card EAM operations. And so I learned not only to run the 7070 and 1401, and to do all the sorting and interpreting and stuff like that

that went with a card shop, but I began to write little 1401 programs. [http://www.multicians.org/thvv/1401s.html] And that was about what I did the first summer. The second summer I did rather more of that because I had my arm in a sling for most of the summer and so I couldn't mount tapes. So they said well, write programs then. Let me just say a little more about Universal Oil because it's interesting. Universal Oil Products didn't drill any wells or pump any gas. Their product was patents, and they had the patent on fluid catalytic cracking, for instance, and a lot of other things. They had vast hallways full of file cabinets full of patent applications. And they also did refinery design and calculation for every oil company, so what we would see is jobs specified on a run sheet with a program deck and then data that might do something called a Fluid Phase Enthalpy calculation, or a pipe stress calculation, or something like that, for building refineries or calculating their projected yield, or working out the economics of building stuff, or of new processes. So that was good because it was practical. I really liked that. That was the other thing that I found as my thinking about careers developed, was that pure mathematics wasn't as interesting to me as something that came out with a concrete answer. And so my third year summer job was at Swift & Company Operations Research Department and I dropped into that; I just found them in the newspaper, I think. And at Swift & Company, the OR Department did a lot of things for the whole company, so once again it would be calculation requests, or can you do this, or stuff like that. Some of the jobs that I remember working on were valuation of Swift's privately owned life insurance company. It was one of the first companies to do life insurance by payroll deduction and they owned their own life insurance company to do that. And if you have a company like that you have to figure out how much it's worth by looking at the population of subscribers, how old they are, what their life expectancy is, and what their pay is, and all that data; so it's a big table lookup kind of operation. You know, it would take about one second now, but in those days, it was a multi-hour run with a lot of tapes. Swift had two 12k 1401s with 12,000 bytes of memory; it was incredible! Universal Oil's 1401 had 1400 bytes. So, we did a lot of interesting things there. One of the things that Swift did was to calculate the formulas for making hot dogs, on their IBM 1620. The many locations of Swift & Company would Teletype in, every Friday, the spot prices for various components of hot dogs. And there were also rules; like in St. Louis, you put more of this in the hot dog and less of that. And there were state laws that said that a hot dog had to have at least 49% beef if it was to be labeled all beef, or something like that. So computing the least cost way to make hot dogs subject to constraints is a classical linear program. Swift set up a linear program and would calculate that out. And actually, most weeks we already had a feasible solution from the previous week and we just had to change the input values a little bit and we could quickly get an answer. But every few weeks we would have to run a longer run to compensate for accumulated round off.

Yost: Very interesting.

Van Vleck: It was fun. But the point was that then I had a grounding. I was lucky because I had a grounding not only in theoretical computer science and in abstract mathematics, but I'd actually seen computers doing things that people wanted done, and that was uncommon at M.I.T. I was often frustrated in talking with people and saying nobody will want that, they will want this, because I had actually seen it happen and seen it done.

Yost: With other students, with faculty members?

Van Vleck: With other students, I think, mostly. And colleagues, as I took part time programming jobs during the school year. And in particular when I was working for Political Science, it was a great help because I had an idea of what a project would be and how to interact with a customer and find out what they wanted, and then present them with results that were actually useful to them, as opposed to saying yes, here's a page of numbers and your answer's in here somewhere, just go down this many rows and go across that many columns. I learned to present the information and to accommodate the program to what the user's likely use of the result would be. So that was fun. Anyway, working with the political science department was great because (pause)

Yost: And this was still an undergrad part time job?

Van Vleck: Yes. And it was great because they had a lot of things that they wanted to do and they were open about how to do it, and they were very smart, and we had a bunch of different projects. The first project that I worked on as an undergraduate was one where we were doing a kind of Monte Carlo simulation of information flow among people in Eastern Europe, and it was called ComCom, for Communist Communications. In those days, behind the Iron Curtain, people didn't know a lot; there wasn't as much openness about how society worked or anything. And various U.S. government people wanted to know how did people in the Communist countries get their information? Who did they hear about news from? Does it spread by word of mouth? Do people listen to Radio Free Europe? Do they listen to the state-sponsored radio? And things like that. So they had interviewed various defectors and they had pages and pages and pages of defector interviews and then people would set up various models of diffusion and communication, and then we would run a Monte Carlo simulation and see if it worked out. And we were able to program a model, based on theories by the lead professors that matched in many ways with peoples' recollections of, say, the Cuban Missile Crisis. And the professor of Political Science, the guy who hired me, was Professor Ithiel de Sola Pool.

Yost: I know that name well.

Van Vleck: Dr. Pool was a wonderful man and very brilliant, and had a lot of ideas, and was quite a gentleman. How did you know him?

Yost: From books that he's written.

Van Vleck: Yes.

Yost: He has written some works that pretty much all historians of technology read.

Van Vleck: The book, *Candidates, Issues, and Strategies*, that was written about the Kennedy election with Sam Popkin. Sam was kind of Dr. Pool's right hand man and I'm still in touch with Sam; he's at UCSD, I think.

Yost: What was your first introduction to CTSS?

Van Vleck: Well, let's see. The undergraduate computing courses were in general not allowed to use CTSS because when I first started at M.I.T. in 1961 it wasn't really running. It was demonstrated running on tape, but it was not really happening. So my first actual use of CTSS was in the fall of 1963 when I had the part time job at Project MAC. [http://www.multicians.org/thvv/7094.html]

Yost: And what did you use the system for?

Van Vleck: Writing programs. In more detail: typing in programs in the MAD language and storing them on disk in CTSS and using the MAD compiler to compile them, and then executing the programs on CTSS.

Yost: And had you taken a course that had taught you MAD language?

Van Vleck: Yes. The sequence of computer courses at M.I.T.; everybody took 6.41 first. Excuse me, we're going to have to stop; I think a neighbor's coming over.
[BREAK]

Yost: Okay, you were going through courses and (pause)

Van Vleck: Okay. So after 6.41, the next course I took was actually a graduate course, which was “Introduction to Artificial Intelligence,” and that was taught by Prof. Marvin Minsky himself. He was a wonderful lecturer; and he brought in outside lecturers to talk about various things; but it was a very broad brush survey course. The field of artificial intelligence at that time was running in all directions and we heard about all of it.

Yost: It must have been fascinating.

Van Vleck: Oh, it was great. It was great, and as I say, Marvin was just a terrific lecturer. And then, the key course to take at M.I.T. if you wanted to be a programmer, was one called 6.251, which was System Programming. That course was designed and written by Corby [Prof. Fernando Corbató] himself, and basically it taught you assembly language programming, and programming in MAD. So that’s where I learned MAD. The keystone of the course was that you took a large program that already existed, an assembler, actually; and you added functions to it. So this Classroom Assembly Program was provided on an archive tape, basically, and your job consisted of merging that archive with alter requests, specified by line number, to delete, insert, or modify lines in the file. And then you would assemble that assembler and then feed it test cases. There were so many points for adding various features to the assembler to make it do more fancy stuff; like you got a whole lot of points by putting in the feature that would allow it to do arithmetic expressions during assembly. So, you know, five times two, or the value of this variable plus seven, or something like that. Word got around to other people about who got an “A” in that course, so were valuable to people looking for undergraduate programmers and they would ask, “did you take 6.251?” and “what grade did you get?” I got an “A” in it; I think I missed a few points at the end of the CAP exercise, but that was kind of the key deflection point. Many of the other courses in programming that were available then at M.I.T. were aimed at specific machines. There was one course I didn’t take because I couldn’t schedule it. It was in the civil engineering systems laboratory, where they had an IBM 1620. But I had learned the 1620 when I was at Swift & Company, so I didn’t really need to learn that machine there [at M.I.T.]. So anyway; so I had learned MAD by taking 6.251; so popping back to the Project MAC job, that was my first real use of CTSS. Now we get a little bit into computer security.

Yost: There were no passwords at that time.

Van Vleck: That’s just what I was going to say. In those days, if I wanted to log into CTSS, I found a terminal and said LOGIN T112 2962 [T1 – twelve – two – nine – six – two]. And it said “WAIT,” “READY,” then I had a session.
One of my colleagues, who also worked for Professor Teager, as a hack one weekend typed in an infinite loop and left it running. And what that did was caused the machine to run that loop whenever there was nothing else to do, instead of running background. You remember CTSS had, in addition to running some number of terminal foreground users, also would run a traditional FORTRAN Monitor System batch background stream with tape input and tape output. That’s why it was called compatible. And background had the

lowest priority of anything. So this infinite loop stopped background from running for the whole weekend and some people were upset on Monday morning when they found that out. And, of course, there was no way to prove who had done it because there were no passwords. So — I won't say "so" — I wasn't there for the decision. But I do know that a few months later, the feature of passwords was added to CTSS and everyone had to have a password.

Yost: Do you know who made that decision?

Van Vleck: I wasn't there to know that first-hand. The people who were relevant were Professor Fano, who was director of Project MAC; and, you know, the story of the genesis of Project MAC, you have (pause)

Yost: ARPA, the $2 million grant.

Van Vleck: Yes, all that. And Professor Fano was the guy who bet his career on computers, if you think about it, because he was one of the lead electrical engineering professors; he'd written the book that everybody who took electrical engineering learned circuit design from. And for him to go off and say well, this computer thing looks interesting, I'm going to basically not do other research in order to do this, was a gamble on his part and it paid off. But Professor Fano was the director of Project MAC, and the assistant director was a fellow named Richard G. Mills, who had worked on Whirlwind, among other things, and a wonderful guy. And he was the one, I think, who actually wrote the code that put passwords on things and also did a bunch of computer resource accounting, time accounting. And this code produced records of what each account had used and quotas were assigned so that if you used more than your limit of computer resources then you wouldn’t be able to log in. Of course, there was a lot of undergraduate distress with that because it’s real easy to use a lot of computer resources by accident or on purpose, and the administration of CTSS felt that they were responsible to NSF and DARPA for the proper use of these resources so they had to have something in there. And they tried to establish that messing with the computers was a no-no. Now, as undergraduate hackers at M.I.T., as you sat around the dormitories and had bull sessions, talked about exploring the steam tunnels and the M.I.T. practice of putting various decorations on M.I.T.’s public facilities, like the Great Dome. And the story was that there were a couple of things you were not to mess with. I was taught this when I was a freshman. You don’t mess with the infirmary in any way: it is not a hacking zone, that’s all. And same with M.I.T.’s power plant: you leave it alone. Interference with either of those facilities would be dealt with harshly. In many other cases, if you did something, obtained a master key, or whatever, and got caught, the administration would say don’t do that again very firmly, you know, and perhaps put something in your record that would later be erased. But there were certain things that were treated severely and the Project MAC administration wanted to make messing with the computers a no-hack area. Didn’t work. It didn’t happen.

Yost: Is that because top administration didn’t support that?

Van Vleck: I don't think it was that, because many of these things were not written policies anywhere. It was a matter of whether there was sufficient will to follow through, and whether the actual damage from any event was such that, you know, the institute really felt wounded. And I don't think that ever really happened. I can remember Dick Mills' finding some people who had done something that they should not have done and saying to me, "Well okay, I have to have another crucifixion today." And taking them into Fano's office and, you know, members of the administration and these terrified undergraduates, and the door would remain closed for a long time. And they came out and they behaved better after that. So that was about the extent of things, nobody was kicked out of school for computer hacking and, in fact, people treated a certain amount of exploration humanely and tried to direct it into positive results rather than simply smashing people who stepped over the line. So, anyway, passwords were a feature and Steven Levy mentions how people in Marvin Minsky's Artificial Intelligence Lab thought this was a terrible idea. They had their own computer, which was much less heavily used than CTSS, and they had a tighter community of people, so they were able to use society pressure to keep people in line. But they viewed the CTSS approach of rules and quotas as being insufferably bureaucratic.

Yost: Was Fernando Corbató also involved early on with CTSS' security initiative?

Van Vleck: He was the leader of CTSS; it was his idea. In 1961, or thereabouts, he and Marge Daggett, and later Bob Daley, were the creators of CTSS and it was the success of CTSS; in fact, there was a lot of desire for time sharing. Various projects, including

Teager's and Jack Dennis, and a bunch of other people talking about how to do it, and Corby, who was deputy director of the computation center, was able to cause a demonstration project to be done in 1961 that demonstrated it could be done; and then was able to upgrade the IBM 709, IBM 7090 to an IBM 7094 with features that would really support CTSS. And that was his leadership. Because that had worked so well, that was the reason that Project MAC was, I mean, that Licklider was impressed and proposed Project MAC; and there's the story about writing the proposal on the train, right?

Yost: Yes. Were you aware of Corby's ideas with regard to passwords and security with CTSS?

Van Vleck: I was not at the time aware of Corby's involvement. I didn't really get to know him until after I had graduated. So, you know, I could go on with hearsay, but that's not necessary.

Yost: Can you talk about efforts to prevent CTSS from crashes and loss of data?

Van Vleck: Sure. Every system crash on CTSS was a distressing event. You would hear cries of anguish as the system crashed; and when it crashed the 7750 terminal controller typed out "CTSS not in operation" on every terminal that was connected. And you learned the rhythm of that sound and knew that something bad had happened. To bring CTSS back up sometimes took a long time, half an hour or so because we had to run a program called the salvager, which looked at the disk files to make sure that everything was okay. I could talk for a week about salvagers. But; so every system crash resulted in a crash dump being taken — a big pile of paper — and then system programmers would dig through the crash dump and look for what the system was doing when it crashed and what it hadn't liked and try to reverse engineer what was going on and change that so it wouldn't crash. There were a lot of different reasons for a crash. Hardware was much less reliable in those days than it is now, so that was an issue. And hardware errors would drive the system into states where we hadn't anticipated and recovering from that was not always possible online. Then there were also logic errors and the discipline of programming was very much in flux in the early 1960s. I mean, learning how to make a program that really worked in a real time fashion was something that everybody on a sequence of lessons learned and I got to see lots of people learn lots of lessons; I learned them myself. There were a lot of different releases of CTSS because CTSS was only instantiated on one, and later two, machines it was easy to make a new version of the supervisor and put it up for users. And so there was a rapid evolution there and because the system programmers were also users of the system there was a tendency to tailor the system to the needs of power users and demanding users. And also, during that time, there was a lot of interest in how to do scheduling of multiple processes demanding resources, especially in a situation where there was more presented demand than the CPU could provide. And Corby is famous, academically, for the Corbató-Greenberger Scheduler, which is a multi-queue exponential scheduler, and there were a lot of releases of CTSS where little scheduler tweaks and changes were made to try and improve responsiveness and still not starve the lowest priority jobs, and cut down on overhead. During that time there were so many algorithm changes to the scheduler that although most of CTSS was written in hand-tuned, hand-bummed assembly code, the scheduler module that contained the logic of scheduling was written in MAD for clarity because Corby realized that there were too many silly programming detail errors when trying to maintain a complex program like that, written in assembly language. And so he dictated that SCDC [the module of the CTSS scheduler that contained the policy algorithm] would be written in MAD.

Yost: Users accustomed to batch environments had not grown accustomed to having much privacy of their data because it was in output bins. With the advent of CTSS, did that shape any increasing expectations for privacy or desire for privacy among users?

Van Vleck: It did, but I think that happened gradually and probably the expectations of privacy and the desire for it really began to surface in the later 1970s. And to some extent, they were driven by the claims for better privacy for Multics. Because remember Multics had been discussed and proposed as an idea as far back as 1964, and the idea of saying well, we'll do this and it will be better. It will be better in many different dimensions. But one of them was definitely privacy and security, and I think that much of the desire for privacy and security was not user driven. It was rather led by the researchers and developers themselves who said that this will be interesting. Now there were cases where the needs of power users and system programmers lead us to perhaps more desire for privacy and security. And, in particular, the need for security of the password file so that no one could look at it and find out passwords. And the need for protection from alteration of the accounting records in order to enforce the record quotas.

Those were two of the major requirements, which led to building the privacy and security features into the system.

Yost: Are you aware of any successful efforts to access the password file in CTSS? Was that a problem?

Van Vleck: Well, there's the famous story that's in Corby's Turing Award lecture, of the time when a person disregarded my instructions and edited the password file with an editor on the Comp Center machine. This was back when there were two CTSS installations. And the operator was editing the message of the day at the same time that the contents of the two files got swapped, and the password file came out online. By that time, I had graduated and I was working for Dick Mills as kind of a junior assistant administrator, and when this happened — I think it happened at five o'clock on a Friday, like all of these things do — I was told about this and enlisted in the damage recovery. And although the Project MAC copy of CTSS was not penetrated, many of the system programmers who had accounts on both machines had the same password on both machines so I had to do a big password changing exercise on the Project MAC machine in order to protect people, because if anyone could've logged into a system programmer's account then they could've planted a Trojan horse or accessed the password file or the accounting records. It would've been a bigger mess than it was. So I was involved in the cleanup for that. There were other penetrations of CTSS, several of them done by system programmers as proofs of concept saying, no, we have to change for instance, memory allocation, so that it's always zeroed when it's given to you rather than containing the results of whatever used to be there in core. And Don Widrig demonstrated at one point, that if you didn't do that, you would end up being able to grab the contents of the password file eventually. There were other ways to penetrate CTSS security and there were a sequence of patches and changes to prevent that. So, yes, I became involved in that kind of thing from the point of view of a junior assistant doing projects for Dick Mills. He asked me to take the usage records that the system produced, and produce a nice-looking report. And then there was another thing that I did for CTSS, which involved — and I'd started on this I think when I was still working nominally for political science — the problem was that in order to make any kind of change to the control files, to add a user, or to change somebody's record quota or something required Dick Mills to log in on his highly privileged account and edit a file and then install it. Well, he had other work to do, so what we wanted to do was to divide the accounts on CTSS into groups and have instead of just one guy able to do it, have each group administered by its own administrator. And there would be one for Electrical Engineering, and one for Political Science, and so on. And then these administrators would use a special purpose program for editing the allocation files, and then what we had was a batch job but not a background batch. It was what was called foreground initiated background and it ran once a day. It collected all of the group administrators' specifications, checked them, merged them together, and installed them. We called that the crank. It also did an accounting report every day, and checked for people who were over quota and flagged their account, and did a lot of other stuff. I was the developer of this facility and Dick Mills was the director, the boss. I did a lot of the programming and then he would look it over, and then we'd install it and see what it did. Working on resource management of

CTSS was my entree into the Multics programming group because they needed the same functions and I was the guy who knew how to do it.

Yost: Were you aware of any discussions early in the planning of Multics where lessons about security and privacy with CTSS fed directly into plans for Multics?

Van Vleck: I would have to say I was a junior member and I wasn't personally there. So a good person to ask would be; well, Corby, first of all; and Jerry Saltzer. I think those are the two people who are forthcoming.

Yost: Those are two people, definitely, on our list to interview. And our institute already has an interview with Corby, a lengthy interview, but one rather short on discussion of security. And so, it's 1965 that you graduated from M.I.T.?

Van Vleck: That's right.

Yost: And did you immediately begin working at M.I.T. after graduating?

Van Vleck: I actually graduated from M.I.T. a term early because I worked extra hard, and so I transitioned to full time on sponsored research staff for political science. So my part time political science job turned into a full time job, and they had money to do that, which ostensibly came from the State Department. But in those days, some of the other organizations within the U.S. government accomplished their purposes by funding things indirectly. There was later a lot of campus unrest about M.I.T. taking money from CIA. As we came to find out, it was probably the case. Anyway, the Political Science department had the money; they wanted stuff done; I knew how to do it. So I started working for Political Science, and they found me a desk over at Project MAC. My colleague Noel Morris and I spent a lot of time exploring CTSS, learning how it worked, writing programs for political scientists that used that knowledge in creative ways. And that was when we developed a bunch of neat things for CTSS, in addition to whatever data analysis and survey projects we were supposed to be doing. [http://www.multicians.org/thvv/mail-history.html]

Yost: Were there other academic departments that had created similar positions to the one you had in political science that were also using CTSS?

Van Vleck: Yes.

Yost: What principal departments were using CTSS?

Van Vleck: One way for me to remember that is for me to visualize the terminal room where Noel and I spent many, many, many hours. There was a guy from the department of Mechanical Engineering who also hung out there a lot, and he went on to be a system programmer, too, later on. And there were a couple of guys from the Electronic Systems Laboratory who were working on compiler technology for computer aided design; and some of them went on to work on the Multics effort and other things like that. There were other departments, as well. Civil Engineering; there were a bunch of people who had similar positions. People came and went; and peoples' intensity of focus on using CTSS to solve their problems came and went. But definitely, this was a pattern at that time at M.I.T. Basically, the economics of this was complex because the computer center charged money, that is, $300/hour for computer CPU time, something like that. But the money was "funny money" in the sense that the accounting behind the scenes would allocate to a department a couple of thousand dollars that was restricted money that couldn't be spent on desks or pencils or anything; only on computer time. So there was a lot of fake money floating around in order to basically, allocate access to scarce resources of computers in a way that the top administration felt was fair and felt accomplished M.I.T.'s purposes. Computers were so horrendously expensive, compared to nowadays, that this was a very important and time consuming and contentious process, managing all this resource allocation.

Yost: Under your *Multicians'* website of Multics, you wrote that, "Corby inspired his whole team with the vision of interactive powerful personal computing," so in essence, a computer utility. The goal was "to empower the individual programmer. Batch processing was still quite young but its proponents were vicious about putting down any newer idea." [7094.html, above] Can you talk a little bit about the critics of time sharing, and was that within M.I.T. as well as outside, or primarily from the outside?

Van Vleck: Oh, it was quite widespread. Anybody who has learned any technique tends to get quite invested in it and believe that anything they don't know about is not as good.

And when computer resources were tremendously expensive, there were those who believed it was a big mistake to spend resources on overhead. And classified as overhead is switching the computer between users, or using disk storage for accounting as opposed to actual computed results, or other stuff like that. So, one criticism of time sharing in the beginning was it was inefficient; that it used up about half the resources of the system just in machinery and overhead, in order to provide computing with the other half. And a time sharing system is no good if it's fully loaded and there's demand for every micro second because it's very hard to make it responsive in that condition. And yet that was the condition in which we were trying to run CTSS and then Multics; there was demand for a lot more than we could supply. Now, you know, we have whole computers that do nothing but sit there and run the screen saver, right? But criticism was also leveled at time sharing because it was sucking up peoples' attention; it was diverting people from fluid phase enthalpy profiles and stuff like that, and then members spending time just building more time sharing. There were those who thought that that was a waste because the things that were invented would become obsolete very quickly, and that the manufacturers such as IBM and Burroughs and the like were a better source for system level innovations because they had more control of hardware/software co-design and could accomplish more. And also that the manufacturers had a lot more resources, so that the idea of M.I.T. developing its own operating system, when we could at most put a few dozen people on that development was laughable when compared to IBM's ranks of programmers. And from our point of view, as system developers, we felt like we could do more, and better, and quicker, as it were, to be more agile — although that term wasn't used then — and CTSS and Multics developers were always very proud of how much we did with only a few people, as opposed to IBM's human wave. I was thinking for some reason about TSS/360 the other day, and they had over 1,000 programmers working on it. They got in each other's way, I think.

Yost: I saw from your resume on LinkedIn that you were at M.I.T. from 1965 to 1974, but in different departments; political science — which you've talked about — the M.I.T. Information Processing Center, and Project MAC. For chronology, did those overlap at all? Can you give the chronology of where you worked?

Van Vleck: Okay. So I started, as I've mentioned, in February of 1965, as a sponsored research staff member working for the political science department. Over time, it became clear that my interests and talent lay in working on CTSS itself rather than applications of it, and an amicable transfer was performed and I worked at Project MAC.

Yost: Do you recall when that was?

Van Vleck: I'm not sure; it would've been 1966 sometime. And then I worked at Project MAC as CTSS system administrator; and then gradually shifted over to working on Multics, as well; and then Multics exclusively. But for a long time; what was happening was that, you remember, by 1965 there were two IBM 7094s at M.I.T., one at Project MAC on the ninth floor of Tech Square; and the other in Building 26 in the jewelbox computer room, operated by the Computation Center. Corby had been Deputy Director of the Computation Center, but he had had to give that position up in order to spend full time working on Multics development in about 1965. And so the Comp Center was run by several other people and they received system distributions from the system developers who almost all had gone with Corby to Project MAC. In early 1965 there was a big push to stabilize CTSS and reduce it to practice, and then get the original developers off that and onto building Multics. That culminated in a major kind of 2.0 release of CTSS in August of 1965, when the CTSS new file system was unleashed, and a lot of other changes were brought together. After that, Comp Center and CTSS were parallel installations and the Project MAC guys were only involved in debugging of CTSS when there was something bad happening that affected Project MAC's ability to develop Multics using it, and development people like Bob Daley and other experts were managed to not spend a lot of time improving CTSS anymore, and instead get on with the next thing. So — how'd we get into this? — (pause)

Yost: The transition of your time from CTSS to Project MAC.

Van Vleck: So I made the same transition but somewhat later. The same general idea that Multics was the future and CTSS was the past. So, for example, when the password file problem occurred, the problem occurred on the Comp Center machine and I was brought in to help recover because I knew how the CTSS password system worked, and I had written the allocation code, which was what the guy who screwed up should have been using instead, but he had taken a shortcut. Comp Center was viewed as more of a production facility and was supposed to be more responsive to a broader class of M.I.T. departments and users. Project MAC's machine was about 60 percent dedicated to

Multics development, and there were some other Project MAC projects. The funding story of Project MAC is one that was very complex, and I think Mills would be the right person to interview on that. I've been meaning to obtain a copy of Kenneth Flamm's book on the financing; where he touches on the financing of Project MAC, and ARPA's involvement. But I haven't tracked it down yet.

Yost: So was it roughly in the 1965-1966 period that you were spending some time with CTSS and some time with Multics?

Van Vleck: Right. So I transitioned from doing survey analysis, and statistics, and databases for political science, to mostly CTSS; and then from mostly CTSS to being the guy who supported the tool that we were using to build Multics; and then started working on features for Multics like resource management, and accounting, and then security, because I was the Sys Admin-kind of guy. I was brought in on a lot of the discussions where people would say well, what about this, and what should we do about that? And I would have the practical operational experience as opposed to the requirements or theory.

Yost: What was your earliest recollection of being involved in discussions regarding the design of Multics security and what was it for?

Van Vleck: Well, I think the Multics development group was very democratic and open and so anybody was welcome to any discussion. I can certainly remember having discussions with people that I have worked with on CTSS features, and talking with them about how the file system would work, and how we would apply the richer permission level to securing various kinds of accounting files. That was, I think, one of the first discussions. And then there was a lot of discussion about using the architectural features of the Multics system to prevent system intrusions and to lock the system up tight. The idea of locking a system up so that a user program couldn't scribble on the supervisor was still one that many contemporary systems hadn't completely figured out and certainly IBM, at the time, was oblivious to the idea that the operating supervisor should be protected from user action of any kind. So this was very definitely a subject where Multics development felt like it knew what was needed and many other systems weren't doing it.

Yost: And would you say that that came directly out of the CTSS experience?

Van Vleck: Certainly a major thread. Now, I didn't know Ted Glaser very well. He is cited as being a person who emphasized the requirements of government security; I didn't hear that first hand. His office was a few doors down from mine and we talked a bunch, but I don't think I was in any meetings where he was the leader of such things. But I was a junior guy. Remember, I was very much on call for nuts and bolts kinds of things; and for when we actually go to do something, how should we do it. Multics, in the 1965-1966 time frame, was still kind of vaporous; partly because the PL/I compiler was not available and we waited a long time to actually get started writing code, and when we did, we wrote perfectly awful code because PL/I was such a difficult language to get hold of. So it wasn't until 1967 or thereabout that we actually began to put together a system that actually computed; that booted up and did things. I was drawn into that effort very deeply because I was asked to help out with the project of getting the hardware to actually bootstrap the system and load it. A group of us worked very hard on what we would now call the system integration task of taking all the pieces and knitting them together, and causing the system to come up. During that time, there were various places where the question would be, can we do this? Or shall we do that? And security would be a consideration at the very low levels, saying no, you can't do that, it would be insecure. So I was very much involved with, you know, kind of the low level and how everything worked, and making that secure; but within a framework of a mainframe computer operated in a trusted facility by trusted button pushers. So.

Yost: Are you aware of one or several of the principals involved driving security, more than others?

Van Vleck: Say more about what you mean by "principals."

Yost: The principal leaders of the Multics team.

Van Vleck: Well, let me think. Corby was the source of leadership and emphasis. All he had to say was "this is important" and his word carried the day. Bob Daley, who had been one of the principal creators of CTSS had become Corby's kind of lead engineer on the direction of the Multics project and made the decision on what features were in and how to do them. Daley was a wonderful programmer. One of the things he was terrific about was simplifying things to their essence. His background is he had started out as a computer operator; and so his background was very practically focused on making the system run and making it not crash, and making it not be interfered with by outsiders. This was his mindset; same as mine. He was very definitely a leader. Jerry Saltzer had been a graduate student who did his thesis on the scheduler, but he was so creative and involved in so many areas, that he was definitely one of the people who was able to take ideas and make them clearer to everyone. His communication ability was especially important. There were a couple of people who visiting Project MAC at various times who were knowledgeable about security. We had Arnold Dumey visit, who had worked at Arlington Hall; I wrote a little story about him, right? [http://www.multicians.org/thvv/dumey.html] There was a fellow named Kit Mercer who was from NSA; he was supplied to us for a while. And there was some other guy, whose name I forgot — it's in the *Multicians* list — who was also supplied to us. [Don Johnson] And these guys, in general, just functioned as programmers. They joined the team; I don't believe they injected requirements or sat in on special meetings where people said well, what about government security and what should we do? There was none of that that I know of. All I remember is that some of these guys were very well-trained about leaving their desks clean when they went home.

Yost: Did security considerations factor into the decision to program Multics in PL/I at all?

Van Vleck: I wasn't there. I don't think so.

Yost: Down the road, can you talk about PL/I and that choice; and did it make it easier to address and prevent certain security threats such as buffer overflows?

Van Vleck: Sure. This is a language subtlety here. I mean, we have now seen 40 years or so of operating system development in the C language. Although Dennis Ritchie and Brian Kernighan were wonderful colleagues and good friends, it has become clear that writing secure code in C language is difficult and if anybody knew how to do it in a reliable way, it would be done by now; and it hasn't been. The C language inherited a lot of its simplicity and power from BCPL [Basic Combined Programming Language]; and in particular, the buffer overflow behavior and the idea that a string was just a pointer came out of BCPL's desire to have everything be just a value that would fit in one machine word. PL/I came from a different direction and string values in PL/I have a defined maximum length, usually, and even if they're declared as kind of variable length there's always a notion of the container they're allocated in and its size, and the current length. And so a string value in PL/I is more than just a pointer, it also has meta data attached that said how big it can get. So it's harder to trick a program written in PL/I into referencing outside the allocated space. You can still do it, especially if you write bad PL/I, but it's also possible to write very tight PL/I in which things are declared to fit in a certain space and the compiler and the compiled code and the runtime support combine to prevent bad things from happening. PL/I does have the problem that most other powerful languages have, of allowing for more than one name to access a particular value in memory, so-called aliasing. And that is usually the key to writing a security breach is to be able to alias something so that it will be used as a pointer value by one part and can be used as a data value by another piece of code, and the data value can then set to point to something it shouldn't. (In FORTRAN we used to do that with COMMON storage.) PL/I provides an incredible number of ways of writing bad code, but when used in a disciplined and careful way you can write PL/I code that's tighter than C code and is harder to trick. There's another detail about the Multics implementation, which is that the Multics stack grows from lower addresses to higher, and many times when you overflow a buffer, what that means is that the storage you destroy is unallocated storage to be used by future calls instead of overwriting previous stack frames, which is what happens with the C language because its stack grows from higher addresses to lower. This reverse direction was probably a good idea when they were first implementing C on the PDP-11, but it was a weakness in the way the language was used. The key thing is that the PL/I compiler is a much more complicated beast and takes much more compiler to compile PL/I than it does to compile C. And the internal intermediate representation of the program, if you will, in PL/I, knows a lot more about the extents and current size of values, and carries that along in its compilation and actually then manifests that in runtime data structures so that you can have a PL/I program, which says I will take arguments of any size, and then when you call that PL/I program with arguments of a specific size the called program then knows what the specific size is; whereas if you do the same thing in C, where you had a C program that said I'll take arguments of any size, and then you called it with something of a specific size, there's no way for the called program to obtain the information of what the allocated size of the thing is, and thus, the called program can be tricked into overflowing a buffer. So that's pretty complicated and I hope I explained it well.

Yost: Yes, it's far clearer to me now. Can you discuss hardware features of Multics and how they contributed to the system's security?

Van Vleck: Certainly. The key idea, of course, was segmented memory. That is, in Multics, the hardware provides two-dimensional addressing. You have a block of locations visible to say, an assembly language program, which is a segment — and it may be segment 300 or something like that — and it has a location zero in that segment, and it goes on up to however big the segment is. And segment 302, say, also has a zero; and goes up to some other maximum size. This is a tremendous convenience to the programmer because then he doesn't have to keep saying oh well, this one comes after that and I have to add an offset of sorts and manage all that information. The hardware's doing that for you. The hardware's doing that with a descriptor segment, and so the convenience of segmentation means that it's another reason why buffer overflows are uncommon in Multics, because when you access off the end of a segment it doesn't jump into the next segment, it gets a fault instead. Also, the descriptor segment contains access control flags that say whether the memory is readable, or read/write, or executable. So one can easily arrange memory so that programs that are being executed can't be accidently stored into. This is a big difference between the IBM 7094 programming environment where common assembly programming idioms stored into the instructions that were about to be executed. With Multics, we had to learn to write in what we called pure procedure, wherein a set of compiled instructions were immutable. The way that the descriptor segment on the segment memory were used, furthermore divided segments into rings and established rules so that less privileged rings couldn't see, or modify, or depend on the operation of inner rings, except through specific gates, which appeared to be just subroutines you called, but when you called them, some magic happened to switch the environment to a new and better defended setup. This was a generalization of the master/slave mode that CTSS ran in, where a user program could similarly do an operation but that caused the computer's CPU to change modes and switch into a protected mode that the user program couldn't see and could only pass arguments to and receive results from. But what rings did was to generalize that into a multi-layered version of the same thing; and hardware supported that in order to make it efficient, although not in the initial Multics. There was a major Multics 2.0, as it were, about 1972, 1973 so we often speak of 645 Multics versus 6180 Multics. The 6180 architecture is the one that had hardware-supported rings. Before that, we simulated them by using segmentation and multiple descriptor segments.

Yost: In Elliot Organick's Multics book he wrote that there was an initial conception where there was a provision for up to 32 concentric rings.

Van Vleck: Elliot's book, first of all, describes only 645 Multics. He wrote it, you know, in 1967, 1968, something like that. And actually the hardware would do I think 64 rings, of which we were going to do 32 for users and 32 for the supervisor. I don't think we ever used more than three. The overhead on ring switching was more than we wished it would be; it was several hundred instructions to make a ring crossing. And there were a lot of other complications, some of them having to do with security of the simulated rings on the 645, so we were really glad to have the chance to redesign the processor and move a lot of that support into hardware. The price of that was to cut down to a maximum of eight rings but that was more than we needed anyway.

Yost: Organick wrote that Multics could protect users both from inept practice mistakes from other users, or self-made mistakes, as well as from foul play.

Van Vleck: Right.

Yost: At M.I.T., how widespread was foul play in the early Multics era, the late 1960s, start of the 1970s?

Van Vleck: Pretty rare. Pretty rare. There may have been undergraduates who attempted to see what they could get away with, but it wasn't a widespread practice. Now, one of the projects I worked on while I was involved in the system integration and boot mode thing was to do quite a bit of work on what was called the Fault Interceptor Module. That's called the FIM, F-I-M. The FIM would be invoked whenever there was a trap of any kind, as it were, and it would've been possible to switch the mode of the CPU execution to a privileged state and do something, take care of an interrupt, or the trap might be a request to cross a ring. So the FIM had to examine the state of the system CPU and make sure that it wasn't being tricked into doing something bad. And so this was very, very complex logic; very dependent on the specification of the CPU, which was still an engineering prototype, in those days. And on the interpretation of the specs written in Phoenix, and then implemented in wire-wrap by computer board designers; and then shipped out to Cambridge and further updated with field changes. So it was real complex. But if we got anything wrong the result would be that someone could store where he should not, or cause a system crash that he shouldn't have been able to cause, or whatever. And there were very few penetration attempts up until Paul Karger and Roger Schell. But there were very few penetration attempts and certainly none very organized and disciplined before that. But there would be occasional crash dumps that we would attempt to solve and we would say, you know, here's a path through things that isn't properly checked and we have to add a few instructions to defend. So that was, as I mentioned before, that was very nuts and bolts, down to the bit implementation of security and we had to have the vision of where we were going, and the vision of saying protect the supervisor from interference, and protect users from each other with the principles laid down by CTSS that expressed themselves later on.

Yost: Bell Labs, until 1969, and GE until the Honeywell acquisition of their computer department the following year, were partners with Multics. Did they have any influence with regard to the development of security features up 'til that 1970 period?

Van Vleck: Well, remember this, the development organization was very democratic and so when specification documents were produced, everyone commented on them. The system administration package that I worked on was certainly one of the control panels, as it were, which enabled us to set security settings and to take the vast complexity available and focus it into a coherent set of simple ideas like users, and accounts, and so on. The initial version of the system administration package was done by Joe Ossanna from Bell Labs, and Mike Spier, who I think was working for Project MAC, but I can't remember whether he was Project MAC or GE. That in itself is instructive, that we didn't really distinguish; we knew where the Bell Labs guys were, because their offices were far away, so we always knew that they were Bell Labs guys. But other people, you didn't really worry about who signed whose paycheck, we were all in it together and working hard on it. I don't think there were any specific Bell Labs or GE security requirements or suggestions or anything like that. It was down to individuals. You know Bob Morris, for instance, was one of the Bell Labs guys, and he was well known as a guy with a sense of humor and a what'll-happen-if-I-do-this kind of attitude. And I know that one of the first times that we actually booted up Multics and had it available for terminal usage in Murray Hill that Bob Morris walked over to a teletype and held down the return key and sent in a whole stream of characters, and Multics crashed. Well, he had found a place where there was a buffer overflow of sorts. Basically, what he found was a place where if he did that, the tremendously slow code that we had then couldn't keep up and eventually it ran out of space and it crashed because input was coming in faster than we could process it; can you imagine that? A system so slow that just the repeat rate of a teletype could crash it? Things were bad at the beginning.

Yost: Multics, along with System Development Corporation's ADEPT-50 were early models for computer security.

Van Vleck: Right.

Yost: Was there any interaction between those two projects, between Clark Weissman and the Multics team?

Van Vleck: First of all, there may have been a bunch of such interaction that I wasn't involved in. I think some of the senior members of the team were much closer together but at one point when I was hanging out in the terminal room, there were some people from the Electronic Systems Laboratory and they, in turn, had visitors from industry and some of those people were from SDC and they were working on the ALGOL Extended for Design language, AED language, which was lead by a guy named Doug Ross at ESL, and Doug had previously done the APT language, which was programmed numerical tools and the like. Anyway, so I got to know some of these SDC people and actually, at one point, visited SDC on the west coast to see my friend. And I may have met Clark; I don't think he was there that day; but I certainly did get to see the Q32 running time sharing. And, naturally, as assistant programmers, we were talking about features of our system and stuff like that. "Oh, I can do this, can you do that?" But there was no organized exchange that I know of. People tended to read each others' papers at conferences and perhaps socialize afterwards, but there wasn't any organized thing that I know of.

Yost: In the early 1970s, Roger Schell and Steve Lipner organized a group of people that Ted Glaser was officially in charge of that became known as the Anderson Committee.

Van Vleck: Right.

Yost: Was there any interaction between that committee's work and M.I.T. and Multics?

Van Vleck: Not that I know of.

Yost: What about following that work, the contract went to MITRE that led to David Bell and Len LaPadula working on a security model and also developing a Multics implementation of that model in a paper that Bell wrote in, I believe, in 1974.

Van Vleck: Right.

Yost: Was there interaction of Bell and LaPadula with you?

Van Vleck: We were aware of those guys. Steve Lipner was a good friend of mine; we were in the same class at M.I.T. and actually lived in the same dormitory. Then later on, he became a neighbor of mine and lived nearby. So we knew of each other's interests and activities, to some extent, and the Multics group also knew of the MITRE group and their attempt to lay down a good mathematical basis that would describe the kind of security the government wanted and see if it would map onto Multics. Primarily also because by then, the GE — I forget what the name of it was, the Special Projects Office or something like that — was trying to sell a Multics to the government. And they did, of course, they sold it first to Rome Air Development Center at Griffiss Air Force Base — and they were trying to sell into the Pentagon. So there was government interest, possibly led by the Hanscom Field crew to using Multics to solve the problems which were identified in the Anderson Report and I think we knew about those guys; there were a few meetings as the Multics group worked together on this. And then as a result of some of the MITRE Hanscom Field work, there was a Homewell-Air Force project established called Project Guardian, which was to make changes to Multics, to the standard product. This was important to make security features a part of the standard implementation of the systems because the government, even back in the early 1970s didn't want to be buying specials. They wanted to buy off the shelf, as it were. And so they wanted the off the shelf products to meet their requirements and the Project Guardian guys were funded by the federal systems operation, that is a marketing branch of whatever it was, GE or Honeywell that day — guess Honeywell, mostly — so they were marketing-funded programmers, though, who worked in the space where the GE, Honeywell, Multics guys were. But they had a specific focus on writing design papers and designs to fold security features into Multics. Wasn't much in the way of security features for quite a while, other than saying we want to be able to identify features of Multics that address specific requirements. And then later on when Project Guardian wanted to attach security level and category to every object in the system, then that became a significant exercise to say well, could we do this and if so, how? And I was involved in a lot of those discussions partly because by that time — first at M.I.T. and then at Honeywell — I was working on a project to redo the file system, the Multics file system. And as part of that we folded in the ideas of level and category on everything. I also had been the lead designer of the system administration package so that meant that I could figure out the hierarchical structure of how a user would be identified with a level and category, and how that would then be propagated down to the actual running system and attached to a particular process. And we had many detailed discussion meetings on how things could work, and how they must not work, in order to satisfy government concerns. For instance, UNIX has the concept of a super user who is immune from access control and he can do anything he wants and it's just not checked. This was something that the Hanscom Field guys passed on as saying no, that is not going to happen, it cannot happen. Similarly in Multics there used to be a switch that said don't check access control right now. And they expressed deep distress about having such a switch and having it checked anywhere. And so we eliminated that. We used to use stuff like that, for instance, to make sure that the incremental backup dumper, when it dumped files to tape, would be able to get at the data. After many discussions we decided no, we want access control to control access, and we will give the dumper the ability, if we really need to, to go in and work its way down from the root of the tree, and give itself access, and keep a list of what it did and then undo it on the way out. But we won't have a switch that says just don't check. So there was stuff like that. It occurs to me that there was one thing that I got sidetracked and didn't finish; when you were asking about my personal involvement we talked about how I went from political science to working at Project MAC. Well then, in the late 1960s, 1969 or thereabouts, I had done a bunch of different projects and was looking for something new to do, and Dick Mills by then had gone on to become the Director of the

Information Processing Center. He said hey, how would you like to come over to the Information Processing Center, which will be the eventual operator of Multics on campus. It would stop being a research computer for Project MAC, which was having its own funding difficulties, and M.I.T. IPC will be running it and CTSS, and another time sharing system on the 360/67, okay? And you know all about that, I hope.

Yost: Yes.

Van Vleck: Okay. And he said, I need some guy to run all three of those. And so I did. And I worked at M.I.T. IPC, so in a sense what I did was I became the customer for Multics.

Yost: And do you recall the years at IPC, was that 1969?

Van Vleck: I think I transitioned over to IPC in 1969. Maybe it was late 1968 but anyway, about then. And then in 1969 was when the Information Processing Center began putting actual "paying" customers, where paying is in quotes, onto Multics, first in the summer of 1969 when it was Political Science; all my buddies from political science who had a project they wanted to do. And that was the initial team that later built what was called the Cambridge Project, and they built a thing called the Consistent System, which was one of the biggest Multics applications. But that summer they were trying to figure out how to program on Multics at all, and one of my duties as an IPC guy was to spend a lot of time with them saying here’s how to use segments, here’s what you have to know about PL/I. And they would do stuff and the system would crash, and I would say oh gosh, that's terrible, here's how to avoid that. Then I would run over and talk with my system programming guys and say, they want to do this, they want to do that, we need a feature for this, we need a feature for that. So, it was a very busy project where, to the extent that it succeeded, I helped a lot; and to the extent that it didn't succeed, I probably should've done more. But then all of that led up to, in the fall of 1969, M.I.T. actually opened Multics up to brave computer users who would use this new operating system. And there were people who were foolish enough that they wanted to do it, or people who saw that this was the future thing and wanted to get in on the ground floor. And so I was very busy, both adding features to Multics system administration to support actual paying customers, and also babysitting those customers and encouraging them.

Yost: Are you aware of any early users that were choosing Multics because of security?

Van Vleck: Well, yes, there were a few. That was where we began to see some Air Force projects and some other government funded people getting accounts on Multics just to explore around and to try things and to see what it was like. There were such users. Most of them were not at all like the political science project that was very demanding in terms of –

Yost: They were from the government, not (pause)?

Van Vleck: Right. There were various government people who had accounts and played a little bit, you would say, but they weren't real sources of input.

Yost: And you were at the IP Center (pause)

Van Vleck: I was at IPC doing various things. We also turned on the time sharing option for the big OS 360 batch system and so I was involved with administration of that, as well.

Yost: And you were there until?

Van Vleck: I was there until 1974, and then at that point, I had been getting more involved with the Multics file system design, mostly because of a remark by the director of IPC, who had followed after Dick Mills, a fellow named Bob Scott. Multics had crashed and I said oh well, Bob, we'll have to come back in a half hour because it takes a long time to get back up. And he said well, how come the CPCMS can be back up in five minutes? Well, good question! And that was one of the things that led us to try to redesign the Multics file system so that it would come back up much faster when it was restarted. That took a long time. But because I'd been working on that, I really wanted to work on it full time so I transitioned over to taking a job at Honeywell.

Yost: Do you want to take a quick break?

Van Vleck: Sure.

[BREAK]

Yost: So you were talking about your transition from M.I.T. to Honeywell.

Van Vleck: Right. So that meant that instead of going to an M.I.T. building every day, I went to a building in Tech Square, kind of two blocks away, and that was about the extent of it. I saw most of the same people and worked on mostly the same stuff.

Yost: Did you have a sense of how Multics was viewed by Clarence Spangle and the top leadership of Honeywell?

Van Vleck: I didn't meet those guys for a long time. They were viewed as unknowable, capricious, dangerous people, which is just a matter of structure, right? Anybody who is three or four levels above you and can cancel your project is viewed the same way.

Yost: Can you talk a bit about the budget devoted to Multics by Honeywell Information Systems?

Van Vleck: I'm not the person to ask about that. There was always fear that we wouldn't be able to do what we wanted to do; there was conflict; and Honeywell really carried us along for a long time without making a lot of money off it.

Yost: What is your recollection of the early Tiger Team efforts to demonstrate vulnerabilities; the Roger Schell, Paul Karger efforts?

Van Vleck: I wrote about this a little bit. [http://www.multicians.org/security.html] I had, as part of my system administration skill, and the fact that I'd written a lot of the system administration package for Multics, I was involved in meetings with the Project Guardian guys, who were part of the Honeywell group. They had a set of offices over in the Cambridge Information Systems Laboratory (CISL, pronounced "sis-el") and I would go to meetings there with those guys. And sometimes Karger and Schell and a couple of other Hanscom Field guys would come and talk with us about how things work, and about what they were doing, and about what we were doing, and so on. And at one of those meetings Paul Karger handed me a little piece of paper and on it was my password to VanVleck.sysadmin.

Well, you know, this should not be. Now, as I wrote in the story there, after the CTSS breach, which exposed the password file, Joe Weizenbaum said well why don't you store a transform of the password in the password file so that if somebody sees it, they won't be able to guess what it is. And he said, you know you could store the square of the password. I said yes, but people can take square roots, Joe. Well, for a variety of reasons, we didn't change that in CTSS. But when it came time to write the password file implementation for Multics, I said gee, you know, we should do that. And I made up an obfuscation, and what I did was I applied some mathematical functions and then masked it with this. I couldn't look at it invert it in my head; so I thought well, good enough; and there's a comment on the code that said I'm not sure this is not invertible, but I don't have time to deal with it anymore today. Well, it turned out it wasn't invertible and what the Tiger Team accomplished when they did it was that first of all, they used a leftover entry point that was part of the 645 Multics supervisor in order to support somebody's thesis. And the thesis code had been removed but the entry point was still there and if you called it with bad arguments, the 645 didn't have the ability to bounds check arguments across a call automatically. Not bounds check, actually, it's ring check. Most pieces of the supervisor had been very carefully studied and if you called a supervisor entry point and said list all things in this directory and put the result here, and if you gave a pointer to something you shouldn't be able to write, in 645 Multics, most of the modules would check that and say oh, shouldn't be able to write that segment, give an error instead. But this thesis entry point, because nobody was using it, and it wasn't anybody's job to be in charge of it, hadn't been checked. And so they invoked that entrypoint with a bad argument and managed to patch the supervisor to give themselves enough privilege to be able to read any file, and they read the password file.

Yost: In your opinion, was that something that Schell and Karger's past experience with M.I.T. and Multics led them to know about that vulnerability?

Van Vleck: Sure. What that did was to shorten the time it took for them to find the problem. The problem was there and they found it, and they didn't find it only because they had been insiders. You know Roger had been a respected member of the development team when he was a grad student and he did a nice piece of work there, and

Paul had been an enthusiastic undergraduate and learned and thought a lot of stuff when he was on the team. But no, they found the hole and there it was. And then once they read the password file, then they inverted my supposedly not invertible transform, and handed me my password. It turned out they had to decide which of 64 values, because I'd thrown away some bits, was the right one. But only one was ASCII; only one was an English word, so there you go. It's too bad, it was a good password.

Yost: Did that lead to any immediate efforts for Multics enhancements, or did that come later?

Van Vleck: Well, let's see; couple things. One is I think we did immediately take out that leftover piece as entry point, just because we should have done that long ago. As I remember, the Tiger Team found three or four other issues. They found a hardware issue. They found some supervisor issues of various kinds. A lot of those issues were relevant to 645 Multics, which was important because Rome Air Development Center had one or was getting one, but we were already deep into development of 6180 Multics on a new hardware base derived from Honeywell engineering instead of GE engineering, I think. And with a new hardware architecture, that is, a number of security features had been migrated from software simulation into hardware, such as the ring protection and evaluation, which had been exceedingly troublesome part of the FIM, which had to be deeply redesigned for 6180 Multics. This is all Mike Schroeder's ideas as far as moving the ring functions into hardware. So a bunch of those changes were kind of yes, they found holes in something that we knew was obsolete and going to be replaced, and the new one won’t have those problems. But we looked at everything. We learned a lot from the Hanscom Field guys about the ideas of how you would be able to trust a piece of software. In particular, some of the discussions there were things that led to Ken Thompson’s lecture "Reflections on Trusting Trust." He said he’d heard about some of these ideas during Multics days. So I don’t think we immediately put in a big effort after the Tiger Team had found things, to patch problems that they’d found. Jerry Saltzer, for a long time, maintained a non-published memo of security holes in Multics and their status, what to do about them or what had been done about them. And I think that is now available on his website. As I remember there were two memos. One was security holes in Multics and how we fixed them, and the other one was even more closely held and it was things we ought to do that we know about. And I think they’re on his site. He and I have corresponded somewhat about this. (See Clark, D. D., Ancillary reports: kernel design project, MAC-TM-87, June 1977, which includes Saltzer, J. H., Repaired Security Bugs in Multics, CSR-RFC-5, Feb 27 1973 <http://mit.edu/saltzer/www/publications/rfc/csr-rfc-005.ocr.pdf>; Saltzer, J. H., P. A. Janson, and D. H. Hunt, Some Multics Security Holes which were Closed by 6180 Hardware, CSR-RFC-46, Jan 28 1974, <http://mit.edu/saltzer/www/publications/rfc/csr-rfc-046.ocr.pdf>; and Saltzer, J. H., and D. H. Hunt, Some Recently Repaired Security Holes of Multics, CSR-RFC-47, Jan 28 1974, <http://mit.edu/saltzer/www/publications/rfc/csr-rfc-047.ocr.pdf>.) (Also see http://web.mit.edu/Saltzer/www/publications/pubs.html)

Yost: You've spoken a little bit about the government and specifically parts of the DoD as customer. Can you talk a little bit about the early customer base for Honeywell Multics and were there a number of commercial companies that were becoming interested as well?

Van Vleck: Sure. In the 1970s, the first Multics outside of M.I.T., GE, Bell Labs, was Rome Air Development Center. And then there were a few other 645 Multics machines shipped. One, in Billerica Massachussets, was used by Honeywell-Bull to support the development of their level 64 machine, and it was actually used via transatlantic phone calls from Paris. And then they installed a 645 Multics in Paris and there may have been one or two other 645s. But then the 6180 generation came along, and the main early customers, besides; I forget when the Air Force Data Services Center was installed but it was pretty early. But the other two that are important to talk about are Ford and General Motors. Ford had Multics twice. It got a Multics in the mid 70s and started to use it, and then lost funding for it and sent it back, and then got it again. General Motors, in the General Motors Information Science Computing organization, or whatever it was, got a Multics in the mid 70s and used it for a lot of different things. Both Ford and GM got to have fairly substantial size Multics installations. And they were demanding customers because they wanted to grow the system in directions that we hadn't. They basically wanted to put a lot more data on the system than previous users; we thought we had a lot of disk at M.I.T. but they wanted to put much more than that. And that was another reason why the new storage system that I was working on became important because we were able to support much larger file storage systems.

Yost: Were there significant security needs at these automobile giants?

Van Vleck: Yes, that's what I'm groping for here. I think neither of them showed any interest at all in government level security; you know, top secret and all that stuff. We would describe it to them and they would say yes, yes, I don't see how we can use that. Their organizations were not nearly so hierarchical and their information sharing processes were much more fluid than the government, which had regulations and structures. One of the things that has to be mentioned somewhere in here is Conway's Law, right? That every organization builds its operating system in its own image. So a rigid hierarchical organization will build a rigid hierarchical operating system and M.I.T. with its multiple semi-autonomous departments had a structure where permissions and authorizations were first, much more complex in structure, and second, were not necessarily hierarchical except that there was some guy who was in charge of running the system and he could do what he wanted. In many ways, that's a reflection of how M.I.T. started out in computing was that some guy who was running the system had to be able to do it. You can trace this through and look at other operating systems and say oh yes, I can see that. Anyway, back to other customers. Well, there were a few academic customers early on, besides M.I.T. I remember the University of Southwestern Louisiana, one of the selling points for why they got a Multics machine was that they wanted one machine that could do both administrative data processing and also student computing. In those days it was felt that anything that the students could touch was unsafe to keep payroll records on, stuff like that. But Honeywell salesmen convinced them that a Multics machine would be okay. But as far as security requirements flowing from customers, except for Project Guardian-like channel for the Air Force requirements, there wasn't much. We were kind of leading the customer rather than they leading us and we had in general a richer and more thought-out security structure than they knew how to use, or knew how to apply to their situation. So we were often saying to them, you know, you could solve your problem by doing this.

Yost: Do you feel it was a marketing advantage?

Van Vleck: We tried to make it a marketing advantage. I don't know if it worked. I don't think so. Security has always been a subject that causes a customer's eyes to glaze over and he'll say yes, yes, okay, but tell me more about the COBOL compiler.

Yost: Was there any kind of user group of systems programmers, of customers, kind of a mini version of SHARE in its early years?

Van Vleck: Yes. There was a GE users group, very strongly modeled on SHARE called the GESUA, GE600 User's Association. And they were like SHARE only about a hundredth the size. When I was at IPC I had gone to a couple of SHARE meetings and then the Multics group asked me to go to a GESUA meeting and give a talk to them about the future of Multics, and I did. And GESUA became the HLSUA, Honeywell Large Systems Users Association. HLSUA, just as SHARE did, had various subcommittees associated with the various operating systems, and when I went to SHARE meetings I had gone to the CP/CMS subcommittee, so at HLSUA meetings there was a Multics subcommittee and we got to know the programmers and systems administrators for the various user sites and, you know, partied with them. Very important.

Yost: Was there any feedback from customers that led to enhanced secured features or fixing bugs?

Van Vleck: Well, when bugs were found they were reported up through the usual Honeywell system support chain. And so that was kind of outside the user group stuff. At user group meetings there would be freewheeling discussion and we would consider their needs and requirements, some of which required to be translated into concepts and language that we could use and respond to. But I don’t think there were significant requirements that originated from the user group. Once again, we were kind of leading the pack and we were pretty certain that we knew what we were going to do, and when people would ask for features we would tend to not listen and respond with what we were going to do instead. You know, that happens.

Yost: You published an article on the workings of the Multics change request board. Can you briefly summarize how that worked?

Van Vleck: Okay. The Multics development process was in many ways, a forerunner of many other agile development processes and focused a great deal on the activities of an individual programmer, as opposed to bureaucracy and validation. One of the key stages of developing a piece of Multics was describing what you were going to do and a review of that change by a committee of elders. When we were first developing Multics in the 1960s, say, 1967, 1968, something like that, we'd just ask Bob Daley how do we do this? Shall we do this? What about this? And he would tell us. Then he left. He went to RCA and we needed a committee of people to replace him because no one person knew it all the way he did. So we had some senior developers and we would make a one-page proposal that said, you know, eliminate this here feature, or add this argument to a command, or whatever. I was part of the group that came to a consensus on how this should be done and some of the things that we decided were important. We decided that every change should be treated in this same way, whether it was a little change or a big change; that there was nothing too trivial. Even if it was just fix this one-line bug, we still went through a change request. We hated the change review meetings. They were awful. They lasted too long, they got very tedious, but there was no other way to do it that we could think of and so every change was given a review. And sometimes everyone in the room would have a big pile of paper, and we would go to request number 14000 and we would say okay, 14000. And people could read, so they would read the thing, and there would be no objection. Everybody would turn the page without even saying a word. That was fine. Sometimes, you'd get to 14003, and I had one colleague who wasn't real vocal and articulate, but everybody would be about to turn the page and she would say, I'm not quite sure what this says here, can somebody explain this? And five people would come up with five different explanations and we would realize that the thing was not very clear. And so we would postpone consideration of that request and find out what it actually said, and have it redrafted, in many cases, and then consider it. And many times when something didn't get approval in that way it turned out to uncover a yawning chasm of understanding and it took months before we could go back and actually do it right. So, the change review process was done; ideally, it was done before code was written at all. In many cases, the code was already written and they were rushing the form through before we could put it in the system. But we didn't put things in the system without an approved MCR unless it was a true emergency, and if we did that, we quickly faked the process in the official terminology, and to catch up to make everything be checked off. And many times when we would do catch-up requests like that, there would be fascinating discussion about how it really should be done and we might approve the change request, but with the side agreement that further changes would be made to bring it in line with other facilities, or to document it correctly, or stuff like that. One of the things that went with a change request was supposed to be marked up draft documentation explaining how it worked; we were trying to force actual documentation at the user level to be thought about and written before programming of the feature. And that sure helped because otherwise, it never would've gotten done. The change request was kind of a focal point but work had to be done ahead of that. The canonical way of doing a system improvement would be to first write a concept memo and then have a set of discussions about that, which might result in more concept memos or revisions or whatever, and then breaking down. So in a sense, those would be requirements or high level design. And then breaking it down into MCR, where you actually said well, we're going to change this, this, this, and this to do something. That would be the next phase and then actually programming the code and having the actual code changes reviewed by a competent colleague was another part of the process. And all of those parts of the process had to be done, none of them were sufficient unto themselves. Doing them all together had the effect of getting the important, relevant members of the programming team very knowledgeable about what the system actually did and how it ought to do it, which in some cases, we knew it didn't and we would be coming as close as we dared.

Yost: In thinking about this overall process, this system, were there any influential models or was this pretty much internally developed?

Van Vleck: It was pretty much internally developed. Charlie Clingen had gone to the NATO software conference back in the 1960s, where the term Software Engineering was first used. And so he was a member of the community of people thinking about process, and there had been various efforts within GE/Honeywell, to think about how to do software engineering, but our process was very much homemade and focused on solving the problems we saw. We didn't spend a lot of time, for instance, trying to adapt ourselves to the engineering drawings' waterfall approach that hardware engineering used and wanted us to adopt, because it just didn't work for us. Instead, we crafted our own thing, which I know they felt in some cases was impossibly sloppy, but it worked.

Yost: In the mid-1970s and the late 1970s, how large was the Multics programming group within Honeywell?

Van Vleck: Well, I would say that probably there were never more than 100 people working on Multics software at any time. Certainly, within Honeywell, the group at CISL might have peaked at about 30 or 35, depending on whether you counted the people at Project Guardian or not. There was a group in Phoenix, somewhat smaller size, perhaps 20 people there; and then there were some Federal Systems Operation people who were specifically doing things involved with the Pentagon. Stuff like that. But I don't think there were ever as many as 100. At any time, though, the actual central core of the system, the development was led by a much smaller group. There were not that many people knowledgeable enough and able enough to do all this; to keep the understanding of the system in their heads and make progress. So we tended to have small groups of people working very fast rather than large groups of people that then needed to be coordinated and kept in step.

Yost: In doing interviews earlier this year with Roger Schell and David Bell, they really seem to emphasize the importance and show complete devotion to this high assurance model of computer security. To what degree did you identify with that, or were you more focused on what could be implemented and satisfy the commercial sector as well?

Van Vleck: Well, I think we accepted the high assurance model as the best practice, as it were. But in many cases, the things we were doing didn't lend themselves to segmentation into just security, or just function; there was a strong component of the Multics development process that was driven by wanting to add new features, for instance, instead of just do what we'd already figured out in a more, and better, and safer way. So every time someone said oh gee, what if we had a command that could do this, then you could do all these cool things. Well, okay, but that doesn't really fit into the assurance model, it’s kind of antithetical to it and there was a lot of focus on features and like that. But the other thing is that at some point, the high assurance model requires really thoroughly well-worked out models, and when you say okay, great, when can I have one of those? The answer was always well, gee, we’re working as hard as we can and we’ll have it for you as soon as it’s ready. And the development group, quite often, when faced with a thing like that would say okay, and we’re going to do something in the meantime.

Yost: In the early 1980s, was there a growing sense within the Honeywell group of criteria for designating security features and certification being important or was that just what later obviously became significant? There was, of course, thinking about that by the early 1980s within the DoD, NSA, and NBS/NIST leasing to the DoD Security Center?

Van Vleck: Now we’re getting to the end of my time as a Multics developer. I think I left the Multics group in September of 1981. So I missed out on most of it. I mean, a bunch of these security guys I didn’t meet until later on in my security career. I didn’t meet Jim Anderson or some of those guys until much later. I worked for a company that Steve Walker started but Steve and I never actually overlapped at that company.

Yost: TIS?

Van Vleck: Yes. TIS was sold to McAfee, and I worked for it when it was McAfee. I knew Steve but not during those years. So I can’t really help with that.

Yost: You left Honeywell in 1981 to join Tandem?

Van Vleck: Correct.

Yost: Can you discuss your job title, principal responsibilities, and what work you did on computer security at Tandem?

Van Vleck: Okay. I left the Multics group with reluctance, but I felt like Honeywell was never going to make anything of it. I also felt like working on mainframe operating systems was kind of like working on nuclear power plants. There was a question of when a system did crash, how many lights went out? And I felt like Tandem's having smaller computers and more of them was a better deal. So that was one of the reasons I left to go work there. When I worked at Tandem, I started out working on the transaction monitoring facility. I was a system developer, or programmer, or whatever — we weren't a lot on titles — and so I worked on the Transaction Monitoring Facility, then I worked on the development of Tandem's next generation operating system, a project that didn't succeed; then I worked on several other things. As I transitioned from one task to another at Tandem there was a time when the Orange Book had just come out and they said to me what do you know about this? I said I know a lot about this, and I know the guy that wrote it. And they said what would it take for us to do this? So I wrote them a memo on that, and then I wrote a couple of other memos about security, you know, as a project. One of them was trying to describe if you wanted to trust a Tandem system, here's a list in order of all the things you would have to trust, including the truck driver who delivered the system was actually employed by the trucking company. (Laughs) You know, all that. And I met with them and contributed ideas to some of the people who worked on the Tandem security implementation, where they tried to build something that would at least get them to C2. And they had a really sharp guy, an IT guy that I worked with, called Bob Baldwin. He's passed away now. Really good guy. We talked about ways of expressing access control that would fit into the Tandem environment and express functions that were important to the Tandem operating system. And these were different, in many cases, from traditional read/write access because they had to be more complex and more powerful. We had nice design, which I don't think was ever fully implemented in the Tandem security project. And that's about it for security at Tandem.

Yost: And then in 1992 you became a principal engineer and security leader for Taligent?

Van Vleck: That's right.

Yost: Can you discuss that briefly?

Van Vleck: Briefly. Taligent was a company that was formed as a joint venture between Apple and IBM; to take an Apple operating system, develop a second generation operating system once again — which was code named "Pink" — they had been working on it for seven years and it still wasn't done, and they got some IBM money to finish it off and to turn it into the next generation Apple operating system and IBM would also get to use it on their machines because they had a couple of platforms that were getting long in the tooth, such as System 38, which was printing money but it was very ancient technology by 1992, and they didn't know how they were ever going to get another system that would make as much money for them. They thought well, gee, if somebody would do an operating system for them that would be just great. So they formed this company and they had a review. Well, now that we've formed the company, what have we got here? And there was no security in Pink at all. And the story is that as as Apple engineers had been developing on it, that they had been reviewed by Jean-Louis Gasseé and they'd said but there's no security in this, Jean. And Gasseé had said, "fuck security." And so they did. So when Taligent was formed, IBM reviewed the Pink system and looked at it and said geez, there's no security. And the head IBM guy said well, hire some guy that knows about security. And he interviewed Karger, and then he interviewed me, and I lived closer, I guess. He might have been better off with Karger. But I was "the guy that knows something about security" and while I was at Taligent I worked on a bunch of things in addition to trying to advocate for security. And what happened at Taligent was that they never did finish Pink. There was a day when Pink was supposed to be available and it wasn't. And IBM had been working on a UNIX microkernel, the so-called Mach industrial kernel, and they wanted Pink to be jacked up and the industrial microkernel put in underneath and use that microkernel as the basis for the future Taligent operating system. And there were many, many, many meetings and arguments and discussions about whether or not to do that, and finally, IBM prevailed. So at that point, I had kind of a sketch of a security design for the old OS, and it was more or less what I knew how to do. That is, a high assurance model with part of the supervisor that would check everything and some rules that were encoded in such a way that you could do things. So it was the Multics/CTSS/UNIX security model of supervisor designs. When I came to adapt my security design to the Mach industrial microkernel I discovered it didn't work because Mach worked by message passing and the whole design was that various Mach tasks, they called them — they were processes, really — but a Mach task would get a message and then it would do something. But there was no way for this task to discover who had sent it a message. There was no indicator at all. And so I went through a long exercise at Taligent with various IBM architecture review boards, trying to convince them that you had to put a number on every task and then unforgeably have that number received by the receiver, whether or not the sender wanted to send it. And after much battling and the invocation of IBM fellows, things like that, we managed to make everyone see that if they would make this tiny little change to Mach then I could get on with my security design and things would be just fine. About the time all of that completed, Taligent decided not to do an operating system at all, and so all of that went for naught.

Yost: Can you tell me about the history and the context for your developing the *Multicians* site?

Van Vleck: Okay. As far back as when I worked at MIT IPC I had had a list of who worked on Multics that I kept for my own enjoyment. And then in about 1993, a Swedish guy had created a Usenet group about Multics, but there was no Frequently Asked

Questions file, and I said well hey, I've got a bunch of information saved about Multics from old days, I'll just type it up into a FAQ file and post it. And I've done that for a long time now. (I'm about to stop; nobody reads Usenet anymore.) But then the World Wide Web came along, and I said you know, gosh, you could have pictures. And the other thing that had happened was that Lynn Wheeler had made me aware of Melinda Varian's wonderful memoir, "VM and the VM community." [http://www.leeandmelindavarian.com/Melinda/neuvm.pdf] And Melinda; I've never met her in person, have you?

Yost: No, I haven't.

Van Vleck: Well, that is a great piece of writing. It inspired me. I said gosh, we could write down some of these stories. I didn't know what to do with them or where. Melinda is married to a Multician, you know that. Her husband Lee worked at Project MAC in the summer of 1966 or 1967. So there's a family connection all around here. So anyway, when the World Wide Web came along I put up a Web server on a Mac in my office at Taligent in 1994. I asked and they said, sure, you can do that. Here's how you do it. And then I started writing Web pages. And wrote some stories down of what I remembered, and solicited some from other people and said hey, this is going to be great, guys, we can all put our stuff together and have it up there. So, it's been 18 years now and it's still growing. I'm still Webmaster, I guess, and editing it.

Yost: It's a very useful and important site.

Van Vleck: Well, I wish every operating system had a comparable site.

Yost: That would be great but there aren't too many as important as Multics.

Van Vleck: Well, you know, but some of that is just a matter of getting the story out. I mean, there's a great site on BTI. Have you seen that one?

Yost: No, I haven't.

Van Vleck: Oh, well look for it. I'll send you a pointer [http://www.btihistory.org/index.html] because BTI Computer Systems is practically unknown, but they did brilliant work. I wouldn't have known about them but I had an office mate at Tandem who had worked there and done good stuff, then went on to do wonderful things at Microsoft. That's Pat Helland. Pat had great stories about BTI so I looked into it, and it turns out that just recently a site has been put up that describes the history of this little computer company on the West Coast and the good things they did, good technology they had. But they didn't make a permanent success, either. But I wish every operating system had a good website. Dartmouth, you know, you would think DTSS would have a rich site, but they've tried a couple times and every time it's kind of fizzled. It really takes somebody who's going to be a little obsessive about keeping it up. You know, people start these sites and do a whole lot of work on it for about six months and it starts to just kind of spoil.

Yost: Right. Finally, are there any topics I didn’t cover that you’d like to comment on?

Van Vleck: Oh gosh. I think we’ve covered pretty much all.

Yost: Well thank you so much, this has been extremely helpful.