Editors: Bob Blakley, bob.blakley@gmail.com | Lorrie Cranor, lorrie@cmu.edu

High Assurance in the Twenty-First Century With Roger Schell

Bob Blakley: Welcome, everybody, to episode three of the IEEE *Over the Rainbow* podcast. I am your host Bob Blakley, and we're here today with one of the founders of the field of information security, Roger Schell. Roger, back in the early 1980s, was one of the leaders of the production of a U.S. government standard called the *Orange Book* [formally titled the *Trusted Computer System Evaluation Criteria*, (*TCSEC*)], which essentially defined the model for a secure computer system. I have a copy of it here. Roger, you'll be happy to see that I used it heavily enough that I had to take the staples out. I think it's probably a good time to be having this conversation because this year is the 50th anniversary of the reference monitor model. Nineteen seventy-two saw the publication of the Anderson report, which introduced the concept, as far as I'm aware, in the public literature—and that'll be 50 years ago this year. And—as I think you're aware, Roger—the title of our podcast is a sort of multilayered pun, and one of the aspects of the pun is a reference to the Rainbow Series, which the *Orange Book* was the inaugural volume of. One of the sly suggestions we were hinting at in the title was that, in some sense, the computer security industry is over the Rainbow Series—over in the sense of sort of wanting to leave it in the past. But I think one of the things that we're going to talk about today is that it's not easy to leave the Orange Book in the past and also possibly not wise. So, with that introduction, Lorrie, I'll hand it to you to start our questions.

Roger Schell.

Lorrie Cranor: Roger, can we start by having you introduce yourself and tell us what you're working on now?

Roger Schell: I'm president of Aesec Corporation and its subsidiary Gemini Computers Inc. Its Gemini Multiprocessing Secure Operating System (GEMSOS) security kernel was one of the early *Orange Book* National Security Agency (NSA)-evaluated, high-assurance (class A1) commercial products. Since acquiring that technology in 2003, Aesec has worked to find ways to apply it to modern environments, things like programmable logic controllers (PLCs) used in industrial control systems that currently have a lot of emphasis. We recently ported the OpenPLC project to GEMSOS for a high-assurance PLC prototype and delivered that in August 2020 to Matt Bishop's University of California, Davis Computer Security Lab. We've also created a Linux interface to GEMSOS and demonstrated that by taking an open source network file service (NFS) and implementing that on top of GEMSOS. We delivered this multidomain, externally accessible NFS, again as a prototype, in March 2013 for a government customer. So, that's the sort of things that we're doing. We recognize that it's scientifically impossible to create a secure system without a trustworthy operating system, so, as a business, we're obviously looking for customers for our trustworthy operating system who would like to create secure systems.

Blakley: Along those lines, Roger, in your writings and speeches over the last 40 years but, in particular lately, you have talked about verifiable security and how that relates to the trustworthiness of systems. And I'm not certain that 100% of our audience is going to be familiar with verifiable security, so could you talk a little bit about it and, in particular, talk about how it differs from most of what we're

Editor's Note: This article is an edited version of an oral interview. The full audio of the interview is available at the *IEEE Security & Privacy* homepage (https://www.computer.org/security) and via major podcast platforms.

Digital Object Identifier 10.1109/MSEC.2022.3159044
Date of current version: 26 May 2022

Copublished by the IEEE Computer and Reliability Societies
1540-7993/22©2022IEEE

doing in the information security space today?

Schell: Traditionally, what verifiable security meant was reflected in the term *verified protection* we took from that title for class A1 in division A of the *TCSEC*. What that has meant to me is basically to verify that a system would enforce the security policy established for the system's information; that was what was being verified. Today, the term is often used in rather aberrant ways and doesn't mean anything like that—one verifies little about the information security policy.

Blakley: If I could follow up on that a little bit, say a few words about what you think people mean by *verifiably secure* when they don't mean that the system can verifiably enforce the policy, and also maybe a little bit about why it's so important for a system to verifiably enforce a carefully specified policy.

Schell: The notion of software verification in a general sense—verification tools and such—predated the *Orange Book*. That has continued as a research area primarily concerned with verifying the functionality of a program. So, when people talk about a verifiable operating system, what they usually mean is that they have some added confidence the functionality is there, and you can take that to various degrees.

The *Orange Book* pointed out for a highly secure system (called *class A1*) the minimum set of things that one should do to have reasonable trust in the system. That, for example, spelled out: I need to verify I've got a representation (what Anderson called a "model") of the policy that is consistent, I have a specification of the operating system's application programming interface (API) that is consistent, and I need to do that in a way which is very precise, using formal verification tools and things like that. And it spells out specifically where those kinds of formal methods are important to verification.

The *Orange Book* also notes, for example, that for future development (what they call *beyond class A1*), one would look to actually verify the code that implemented that verifiable specification (but not for class A1) and similarly potentially verify the hardware. Those are mentioned as things that were considered reasonable research goals for the future, but all of them focused around verifying that I had access control policy that was enforced by the system, not just verifying the program for sake of verifying the program.

Cranor: Let's talk about another term you've written about: the cyberdefense triad. Could you describe that for our audience?

Schell: I first introduced it in the classroom at the University of Southern California, where I was on the faculty for several years. My November 2016 invited *Communications of the ACM* (*CACM*) piece summarized three things—and you have to do all three in order—to provide "dramatically more trustworthy cybersecurity." You have to have a fundamental representation, of the system—a reference monitor. You have to mitigate subversion of the system, as well. I took the principles from the Anderson report, which was mentioned by Bob, and mapped those three principles to "three legs of a stool" illustrating the triad.

Blakley: I think Lorrie's going to get back to more conversation about subversion in a minute, but I wanted to talk about the third leg of the triad as you describe it, which is mandatory access control. I think when a lot of people in the field think about mandatory access control, they instinctively go to the Bell and LaPadula model for confidentiality protection using mandatory access control. But I think, if I'm understanding what you've written recently correctly, you're more focused these days on what you are calling "integrity mandatory access control," which I guess means an integrity-based lattice model, like the Biba model or something. Am I right about that, and would you like to say anything about it?

Schell: You're basically right. Bell and LaPadula is one model which one could to use to represent mandatory access control, which of course, at the time, was called *nondiscretionary access control*. Saltzer and Schroeder, in their 1975 *Proceedings of the IEEE* paper about security, talked about nondiscretionary policy. They reflected that all access control policies can be partitioned into discretionary and nondiscretionary; there's not a continuum, there's a partitioning of policies between them. Both were represented by Bell and LaPadula. And, if you look at the model carefully in Bell and LaPadula, it's a mathematical model. It actually doesn't interpret the terms, so whether it's confidentiality or integrity it doesn't really say.

I think you used the term *the Biba model*—there isn't a Biba model. If you go look at the Biba paper, it doesn't claim to be a Biba model. It claims to be an interpretation of Bell and LaPadula in terms of integrity. And it came about when the need was there to provide certain functionality. And at the root, for example, even during the Anderson panel, there was some discussion of "well, what really are we talking about in terms of access control?" The observation was, pick up a memory chip. What are the operations you can perform on a memory chip? It has basically a read cycle and a write cycle, full stop. So that was the basis of what we mean by access. Either read or write access comes from the basic technology we have in the memory chip. Also, at the time the Anderson report was written, it was responding to the Ware report, which defined the problem but not the solution. And it couched its problem

statement almost entirely in terms of confidentiality.

However, when it came to the first people after Anderson saying "I really need to deploy something that is highly secure," it turned out the first operational need we encountered was not for confidentiality. It was for integrity, specifically in the Strategic Air Command Digital Network (SACDIN) Minuteman missile control system. The Biba interpretation was created specifically as part of our response to say, "In order to have this understandable in a way which people would believe in it for the control system, we will couch it in terms of mandatory integrity." And that was the origin of the Biba interpretation.

Blakley: Just to follow up on that a little bit, I guess that in a lot of deployments today we're probably in that same situation as we were in the Minuteman days. If we're talking about PLCs and embedded controllers and Internet of Things devices and what have you, we probably are in a situation where we're looking more to concern ourselves about the integrity properties than about the confidentiality properties, at least at the base layer of the system today. Do you agree or disagree?

Schell: I don't totally agree. I think that both are important, and they have different environments. For example, I claim the Equifax breach and a lot of other massive data breaches are primarily concerned about confidentiality. And yet certainly, our recent attention to control systems, including PLCs, is about integrity. So, they're both equally important; they each have their customers that are concerned about them. The good news is that the same basic technology exists and can be applied to both. In the case of our GEMSOS product, produced by Gemini originally, if you look at the June 1995 final evaluation report by NSA, they point out that the security kernel doesn't know about confidentiality or integrity. It simply knows about enforcing mandatory access control policies for a lattice, and whether it is integrity or confidentiality is syntactic sugaring added outside the kernel.

Cranor: So, getting back to subversion, can you say a bit about subversion and insider attacks? How important are they, and how vulnerable are we at the moment?

Schell: You've seen my perspective on that in the November 2016 *CACM* and elsewhere, which is basically starting with the Anderson report, and prior to that, the primary threat that we faced at the national level was subversion. The military understood subversion early on. My early work in the security area, long before the Anderson report, was, in fact, on a system which was handling highly sensitive data. What the military did in those cases was to actually control the production of special computers—everything, hardware and software, from essentially the ground up. Probably the most widely known example was cryptographic devices. Cryptographic devices were built by a special community with highly cleared people, subject to lots of constraints. There's really just one reason why you did it—just one. That's subversion. The problem that you were concerned with is if you went out and bought a computer off the shelf, even in the 1960s, you had to assume that it was subverted, and there were good and valid reasons why that was a proper concern.

So, the military understood subversion as the primary threat long before many other people did. On the other hand, many (most) installations used dominantly commercial items. Even at the level of Department of Defense (DOD) policy for installations that were not highly sensitive, when I would have discussions of the subversion topic in the 1970s, the most common reaction I got was that "nobody would ever do that." Occasionally, I found my perspective implicitly mocked. I remember one of the major, well-known government research agencies not particularly supportive of the high-assurance effort in the Air Force. Their speaker got up in front of a rather large audience and made the point, "We're worried about real people attacking real systems. We're going to live in the real world; we want something that is acceptable risk." He asserted, "In the real world, we're not facing Massachusetts Institute of Technology (MIT) Ph.D.-in-computer-science-graduates attacking our system; that's not what we're worried about." He recognized I was in the audience, so that was a thinly veiled poke. Subversion has been around for a long time, but it hasn't been considered important to a very broad audience until more recently, where networks and other advances made it more practical for the attackers to use, as illustrated by Stuxnet, Solar Winds, and Log4j, stunning (to many) supply chain attacks.

Blakley: So, you alluded there to high-assurance computing, and I think Lorrie's next question is going to be about that, but before we get on to that, I wanted to go back to your earlier conversation about category A1 systems from the *TCSEC*. I think, to the extent that that people still think about A1 systems outside of a specialized community, they often think, "Oh well, those weren't really built, or they weren't really sold and used." So, I think a lot of people today maybe don't have a full appreciation of the history of A1 systems and just how effective they have been against attacks, and I know you've written and spoken about this and about how surprised people are about avoidance of vulnerabilities and attacks in A1 systems, so I wanted to give you the chance to recount a little of that for us here.

Schell: From the government or military point of view, it was only really

high-assurance systems that were particularly important. Even in terms of the *Orange Book*, look at the way it is organized: it has three divisions. Those three divisions were put there specifically to reflect different classes of threat. Division C said, "You're facing the normal casual hacker"; it is essentially the environment that most of the world lives in today. It is one in which they could be interested in division C.

Division B said, "Well, you have situations in which somebody may actually be making attacks via subversion or otherwise from within the system." Lorrie referred earlier to the question of insider threat. Division B was really the first division to deal with insider threat. I mean, after all, the so-called military multilevel security (MLS) is all about insider threat. It's really only about insider threat. It says, "Assume I have on my system more than one domain of security, and I have insiders meeting there on the system, but they're not all authorized to all the information." That's basically what the insider threat is about. MLS was about insider threat. So, division B says, "Yeah, I got insider threat. I need some kind of MLS since I need a way to separate these security domains." But division B assumed that, by some magic or otherwise, you provided a suitable platform. So, you regarded the operating system you ran on as not subverted, simply as a statement of the context that you were in. And you could do various things, as the crypto world did, to make subversion less likely to happen. They built all the software and the operating system; they built all the hardware they ran on. Then they said, "OK, I'll regard that as sort of static."

In contrast, division A said, "Well, the reason we're here at that point in time, in the 1970s, is that we recognize it's not practical for the government to build all the computers that they're going to use." I mean, there was little other reason. If you look at the basic October 1982 DOD Directive 5215.1 that established the Computer Security Center (its charter), it was charged with creating the *Orange Book* for evaluating the security of "industry-developed trusted computer systems." The directive reflected explicitly that it was because DOD components were going to have to procure commercial systems. And when we're going to buy industry-developed systems, we can no longer live in this fantasy world in which we say, "Nobody subverted that operating system." So, division A moved to the case in which you recognize that the trusted system is now subject to subversion—the total platform which you are on. That is why you had the three divisions.

It was largely the military that lived with those situations where security was considered very important, as addressed by using government-built components like the operating system. At the point in history of the Anderson report, they were looking at commercial systems, and you could no longer make those assumptions. That led to the need for division A. I got drafted into a position as founding deputy director of the NSA Computer Security Evaluation Center when the *Orange Book* was being put together. One of my first policy encounters was when I asked, "Why are we doing anything other than just looking at where the problem is?" I was referring to what became known as division A. That resulted in a little sit-down with managers from the Pentagon who explained to me that having only division A is not a politically correct position. You can't do that when you're trying to talk to commercial computer manufacturers, and we have to have these. So, the way the *Orange Book* was literally constructed was you first constructed division A and its class A1, and then you ask, "OK, what are the requirements I can drop to make it more palatable for the lower divisions?" You didn't have to answer the question "What is this really dealing with in terms of the threat?" The question for lower classes is, "How can I make this look more like commercial systems?" When it got to the lowest, class C1, you were dealing with basically any commercial operating system that was out there. So, the full range of classes was created by repeatedly selectively removing requirements from the next higher class.

Blakley: One other thing that I wanted you to bring up is, as I understand it, there are A1 systems that literally operated in production for years without the discovery of a single vulnerability, which I think a lot of people would be shocked to hear.

Schell: In October 2013, I was asked to participate in the annual NSA Cryptologic History Symposium, on a "Roundtable on Cyberhistory" panel, since cybersecurity is needed for cryptography. By 2013, of course, the *Orange Book* was largely gone, and few cared much about it or knew much about it. I presented a slide on the "Legacy of Class A1: Worked Examples," where I listed a half a dozen class A1 security kernel-based systems that had been operationally deployed. As I mentioned earlier, the first one of these designed to meet class A1, although the term didn't exist then, was the SACDIN system for Minuteman missile control. Its operating system was designed to meet requirements for a class A1 security kernel. SACDIN was initially fielded in 1988, and has only been shut down within the last decade. It only shut down because they couldn't buy parts to run it anymore. Over the years, I've watched with some amusement the way people talked about it in the press. There was a May 2016 article in the *Washington Post* that opined that a "major reason" that it was such a secure system was because it used obsolete, low-tech, 8-in floppy disks. A decade or so ago, I was in a Minuteman missile hole, and, sure enough, there was the SACDIN system. I, of course, talked several times to the people at the U.S. Strategic Command, which inherited it, and

part of the difficulty is that the people that are there today have no idea what to ask for, and so the system they built to replace it, in my view, is highly likely to be much less secure than the one they had in SACDIN because they aren't imposing the class A1 security kernel requirements that would get them there.

Another "worked example" I presented was the BLACKER cryptographic system, an effort to use cryptography in a packet-switched network. It had been worked on for a couple of decades or more, and they had all the pieces to provide cryptographic security over a packet-switched network except the problem of a trustworthy operating system. Several times during BLACKER they finally got down to "almost there!" but just couldn't quite get where they'd be willing to trust their system to this operating system.

At the end of the day, under pressure to meet an urgent need, they decided to produce and field BLACKER, but it turned out they still faced the problem. They gave a major aerospace company the job of producing highly secure (NSA type 1) packet-switched cryptographic devices. As I understand it (I was not there), the company later came to the government and said, "Well, we've got two versions here; we'd like know which you'd like to use. On one hand, we've got this platform based on a very specialized operating system that we think meets your security requirements, but its performance is not anywhere close to what you're asking for. On the other hand, we have this real-time UNIX-based platform that we think meets performance requirements, but if you really want your security requirements, it's not really in the ballpark. Which do you want?" Well, if you're a program manager for a critical program that's been a couple of decades in the making and always stumbled at about this same place (high-assurance security), what are you going to do at this point?

What they did is say, "Well, at this point, a number of years after BLACKER started, NSA is in the process of evaluating commercial trusted operating systems. We don't look at their performance; we have no idea what it is. But we do know it's secure. There is a commercial class A1 system that we think can meet your security requirements, and you can decide whether or not its performance meets your requirements." Now, contrast that to decades of history in which they only use software for cryptography that was built by government-cleared people in cleared facilities. These commercial trusted systems were built in an uncleared, uncontrolled environment by uncleared people, and now you're saying, "You want us to look at that?" And NSA said, "That's what class A1 is about! It's about letting you do that." And they did. It turned out that was the only viable choice they had. So, as summarized in the Clark Weissman May 1992 IEEE Symposium paper, both the access control and the key distribution devices for the BLACKER packet-switched network were built on a commercial class A1 operating system. It was fielded as an NSA type 1 crypto product, and the January 1992 memo from the director of NSA certified "that the A-1 assurance requirements specified for BLACKER have been met."

Blakley: That's a perfect transition. I think the answer to "How did they do that?" is "Assurance," and I think that's Lorrie's next question.

Cranor: You mentioned high assurance. Perhaps you could talk a bit more about assurance and tell us what it is and why it's important.

Schell: The way I might say it informally in the classroom is that a high-assurance system does the trustworthy functions it's supposed to do and nothing else. Assurance is the confidence that it does nothing else. In contrast, if you look at a so-called verifiable operating system that's out there today, it may ensure that it does the things is supposed to do. It has little assurance that it does nothing else. That is, in general, an unsolvable problem. And when the Anderson report put together this concept of a high-assurance security kernel, we said, "How can we possibly do this?" Of course, the initial reaction is, you can't! It's Turing noncomputable. You can't say it does nothing else; just impossible to do. Ted Glaser, who was chairman of the Anderson panel, disagreed. Ted was previously part of the Ware panel, worked in Burroughs Engineering, and was part of the Multics group at MIT. He said, "Well, you know, the way we address these problems of noncomputability is to find a restricted way of doing what we're trying to do, and to know we've succeeded, we need a model." But we don't have a model of what we mean by assuring the enforcement of the security access control policy, so he says, "That's the first step." As they discussed it in the group, and with various people outside, they said, "We have no reason to believe such a model is possible," but if you look at the Anderson report, the first step on their road map was to develop a "model" of what it means to have a secure operating system enforce the access control policy.

Since we were considering Multics as a candidate for following that road map, we said, "Let's see if we can have Multics, from a functionality point of view, enforce the MLS policy that we need and if we can create a model that shows that it meets the policy." We, in a joint project with DARPA, had access control functions added for nondiscretionary (i.e., mandatory) access control as part of the Multics commercial product. If we could, obviously, that would meet military MLS requirements. Since such a model had never before been done and was a critical first step to reduce risk. In the Air Force, we went on two parallel paths.

We had MITRE doing one model, which became widely known as Bell and LaPadula. At Case Western University, where Ted Glaser was chair, we had him doing the other model. We developed a model of what the MLS operating system would look like, so abstractly we knew it's secure, and with that we could deal with verification in a computable fashion. We can prove that, even though with just a model we can't prove the implementation is secure. An essential property of high-assurance systems that you asked about is that this model has a formal mathematical proof that the access control policy is enforced. It proves that for this operating system interface, for all possible programs that you could run, there can never be information flow in violation of the MLS policy. That's the answer to your noncomputable question, and that property is what was meant by *high assurance*.

So now, we had a precise model of a high-assurance operating system. If you can somehow show that what you actually built is represented by that model, then you have a secure system. You've got the proof of the Bell and LaPadula "basic security theorem." The next step is to confirm that the abstract state machine of Bell and LaPadula is something that looks like an operating system, e.g., Multics. An operating system has an API. That led to the need for a specification that became known, in the *Orange Book*, as a formal top-level specification (FTLS), and you had to show that that specification was, in fact, a valid interpretation of the model. Then, you could say the FTLS, if it was faithfully implemented, was secure for all possible programs. But, of course, I still haven't built it. I'm still left with the problem that I have to correctly write the code.

I introduced the security kernel notion in a position paper for a June 1972 IEEE workshop chaired by Stockton Gaines. I proposed "a compact security 'kernel' such that an antagonist could provide the remainder of the system without compromising the protection provided." And the reaction was, "Yeah, that's a pipe dream for the obvious reasons; it's noncomputable." So, in the summary of that workshop, at the August 1972 ACM annual conference, there was no mention of the security kernel. The question was: With this abstract specification of the operating system, how can you build one that has high assurance? You have to build it by construction. You have to construct it so that it can be shown to be a valid interpretation.

Edsger Dijkstra was at MIT as a visiting professor while I was a student. His office was a couple doors down from mine, and I'd go have some discussions with him. One of the points he made was that you can't test to show software has no bugs. So, he says, "I'll give you the problem of the 72-bit adder. I give you this black box and ask you to show me whether or not the black box is a high-assurance 72-bit adder. I'll bet my house against your house that you can't tell me. If you say it's not, to win, you have to show me a test case where it's not accurate. When, after extensive testing, you assure me it accurately does 72-bit adding, I will show you that it doesn't." What I do is build into the black box some circuitry that says when two particular numbers (known only to me) are added, I'll produce a wrong answer. That's what we mean by subversion. It fails even though all the tests say it does what it's supposed to do. But tests can't prove it does nothing else; how can I prevent that? We do it by construction. The way we build 72-bit adders is to combine 72 1-bit adders with a carry according to a proven abstract specification for adders.

What can I learn from that 72-bit adder? For a verified design, I have to build essentially a proof sketch of the operating system implementation to show that it is a valid interpretation of the FTLS and nothing else. There are mature software engineering techniques that we found were useful. The David Parnas May 1972 *CACM* paper and the June 1973 William Price Ph.D. thesis at Carnegie Mellon defined the notion of an "information hiding" module whose information is hidden from the rest of the modules. By constructing the operating system using such separate modules, I can, in fact, build a proof sketch. As in mathematics, the theorem for a module can build on corollaries and lemmas below it. That theorem becomes a lemma for the next layer of the proof. So, one of the things class A1 requires is that the operating system must be expressed at its API specification in Parnas's form: inputs, outputs, exceptions, and effects (state). That is required for class A1 because it's the only way we know how to go about making that kind of proof sketch. But, even if my interface meets the Parnas specification, if I build a monolith inside it, I can't get there. So, class A1 also requires modularity inside. You must have multiple independent modules and must have layering.

Paul Karger's May 1990 IEEE symposium paper on the Virtual Access Extension (VAX) security kernel designed to meet class A1 described the layering of Parnas modules. You could conceptually build each layer and run it as a mini microkernel itself; then, you build the next layer on top of it and the next layer on top of that. When we tried to apply that to commercial operating systems, such as Multics, it turns out that's not the way anybody builds an operating system. We simply didn't know how to build an operating system that way. As part of the joint Air Force/DARPA research project, a September 1976 Ph.D. dissertation by Philippe Janson, at MIT, addressed how to go about layering an operating system in a way that is practical. He solved the nub of the hard problem of doing that, so layering is also required for class A1. Then, the question was: You have all these techniques, can you really do it? Following the Anderson road map, as the next step was, we had MITRE actually construct a running security kernel proof

of concept for what was later termed a *class A1* operating system using those techniques. As reflected in the May 1975 report on the PDP-11/45 security kernel, by Lee Schiller, we concluded, "Yes. We can."

That illustrates how, by high assurance, we mean security by construction to meet a proven model. That's a long answer, but maybe that gives you some flavor for why, at the end of the day, it's not a surprise that it is unlikely to ever need patches. In today's vernacular, it is unlikely to have a zero-day vulnerability. It's what you can reasonably expect. If you look at it from a practical point of view and you're NSA in the 1980s, you say, "What is your most sensitive information?" Well, it's your crypto keys, right? Now, in order to build cryptography for a packet-switched network, I must have the power of a computer; I can't just build my type 1 crypto device in hardware anymore—it's too complex. You're asking me to trust my crypto keys to a software operating system? That's never been done, and anyone who would propose that would be thrown out. At the end of the day, they chose a commercial operating system protecting their crypto keys, but only because NSA evaluated it to confirm it met the class A1 requirements.

Blakley: And it did work.

Schell: It did work. It was a type 1 crypto product. It got distributed around the world.

Blakley: Distributed around the world to handle the most sensitive function that the most secretive government intelligence agency handled?

Schell: Yes, by the United States and friendly nations.

Blakley: So, number 1, it was a long answer, but I don't think I've ever heard anybody explain it that way, and that, all by itself, makes me happy that we did this. But it's also the perfect intro to my next question, which—really, I reached out to you for this—which is we started in 1972 with this idea that was, at some level, kind of implausible, and a whole bunch of smart people figured out that there was, in fact, a way to do it, and they did it, and it worked. And we stopped doing it, and we don't use the technique anymore.

Schell: Yes, we don't use the security kernel technique anymore, it's fair to say.

Blakley: The question, obviously, is: What went wrong? And I guess my summary at this point is that the *Orange Book* model failed for what I would call economic reasons, but everything else also failed, except it failed for technical reasons, and it's pretty plausible, at this point, that the technical reasons that have caused everything else to fail really can't be fixed.

Schell: Yes, they are noncomputable.

Blakley: So, I guess the question is: Does going forward, at this point, also mean to some extent going back, and do we have to just accept that high-assurance systems are expensive but necessary? And what do you see the future as?

Schell: Well, yeah, the question is why it failed. As I've said to several people, you know, I'm not a sociologist, and I can't tell you why other people do what they do. But let me observe that I did have an interview several years ago in which a question came up of why do people say we aren't using this. Although I can't personally speak to this, I have identified some of the reasons people put forth, which may be worth noting (this was an interview Luke Muehlhauser, of the Machine Intelligence Research Institute, published in June 2014). Near the top of the list was, in fact, government actions. I mentioned earlier that Digital Equipment Corp. (DEC) created what I think would have been a class A1 virtual machine monitor for VAX. Do you remember one significant reason they gave for why they didn't ship it?

Blakley: My memory on this is hazy, but I think one of the reasons they gave was that they thought that the market was going to be really small.

Schell: Consistent with that, a major reason given in the May 1990 Karger paper was, "The export controls imposed on A1 systems can seriously reduce the potential market." If you're a global company like DEC, you don't offer a product that you can't export; not only is the market smaller, but it's just business suicide to do that. That problem is one which will still loom. If I'm a class A1 PLC manufacturer and I want to sell to the U.S. power grid, that probably is not a problem to me. But there's a lot of other markets I'd like to address. That's a challenge. Now, it is, in my view, a rather significant abuse of government export provisions, and I have sat on both sides of that export decision process. The nominal story is you've got technology that's "dual use," i.e., it's technology we're using, and we don't want other people to be able to use it. Well, I say, on the face of it, as you just pointed out, we're not using it. When I talked to the people who inform those decisions, a decade ago, they denigrated the security kernel as, "That's 30-year-old technology. We can't possibly be interested in using it." How could that possibly be subject to dual use export control?

Blakley: I think we've had a few crypto policy discussions on the podcast already, and it's likely that we'll have more, and I think you already know that I agree with you that those policies have enormously distorted the security market in multiple ways.

Schell: And it was clearly, in my view, not the intended purpose. The second issue that people point out is the so-called equities problem. Twenty

years ago, I couldn't have uttered those words publicly. Today, it has come out, but the way you hear about it, mostly in the press today, is for a cybervulnerability that the government knows about. They may choose, as a matter of "equities," not to fix it or tell the manufacturer because they're exploiting the flaw in their activities. And there's a formal Equities Review Board set up to decide whether a vulnerability should be fixed. Additionally, one of the challenges of a class A1 product is it's built so a third party could evaluate it. It's worse than the usual equity. It's not just that you didn't fix the problem; it's that somebody else can systematically evaluate it to find the problem.

A third issue which people pointed out is that some parts of the government have an inclination toward a monopoly, and they don't want to have security controls that aren't by, for, and of them. You may recall that David Bell, in a December 2006 paper for the Annual Computer Security Applications Conference (ACSAC), addressed a report that "virtually overnight, NSA put all the class A1 vendors out of business." In the October 1982 secretary of defense charter for the NSA center, he explicitly directed, "It is DOD policy to encourage easy availability of trusted computer systems." The objective was to stimulate the market, but those three examples of government actions seem contrary to that directive.

Blakley: And actually, that is a perfect transition, I think, to Lorrie's next question.

Cranor: I guess you've pointed to a few reasons as to why we don't have high-assurance systems, and a lot of them have to do with regulation and economic incentives. What can we do about it, and how can we actually create a market so that we can have effective high-assurance, secure systems?

Schell: Of course, Bob is more of the expert in marketing than I am. I do agree with his hypothesis that it is basically a problem of marketing. I've interacted with major players out there who give advice to industry, and we've tried to sell the idea of high assurance. I had a senior member of one of these big companies say, "You know, you've been a technology success, and a market failure." He continued, "I'm not going to recommend to my customers anything that's a market failure." That's a challenge. That sems a knee-jerk reaction some people have, and since I'm not a marketing guy, I don't know the validity. You know, some people have observed rhetorically, "Do you think there'd be seat belts in cars if it were just left to market forces?"

Blakley: Well, right, and that's actually the perfect transition to the question that I was going to ask to follow this up. If you have something that's a technical success but a market failure, that is literally the economist's definition of a market failure, right? The market is not functioning in such a way that it produces an outcome that everybody agrees would be the desirable outcome. And the canonical ways that you solve that problem, a market failure problem, are either via regulation or via liability. So basically, either you as the government say "This decision is not going to be subject to market forces because we're just going to tell you that you have to do it," or you say, "Hey, we understand market forces, and we're going to create a market force that says if you screw this one up and it hurts somebody, we're going to give you an enormous penalty, so that we recalibrate your thinking." Is that sort of where you think we're going on this one if we're going to be a success, or are there other options, or is it imponderable?

Schell: I think there are other options, but I think the first step is getting to have that discussion about trustworthy operating systems with decision makers that matter. The reality is that we've repeatedly gone to senior government agencies mentioned as responsible in the cybersecurity executive orders and such, and they talked about things like a public–private partnership. I have suggested, "Well, why don't you go out and have a PLC manufacturer build a prototype high-assurance PLC, controlling some critical cyberphysical components in the power grid? And to do that, require that they will build it on a trustworthy operating system designed to meet class A1?" I've been told that you can't say that, you can't say "class A1." Within the past few months, one senior technologist forcefully "recommended steering clear of any references to A1 or the *TCSEC*." The government will not have that conversation, simply will not have it.

My question is: If you don't want to say "class A1," what is your specification? If you're going to have a public–private partnership, you must have a codified specification to put on contract that a private partner can build to. The *Orange Book* was designed as an engineering specification you can put on contract. No answer. What you get is something like the December 2018 Office of Management and Budget policy memorandum M-19-03 that says to "include requirements for developers, manufacturers, and vendors to employ systems security and privacy engineering concepts and methods" provided by "appendices F and G in National Institute of Standards and Technology (NIST) Special Publication 800-160" dated November 2016. Although this has well-written general conceptual, philosophical guidelines of a reference monitor, it is not at all an engineering specification for contracting a manufacturer to build to.

Blakley: Well, anybody could build to it by using the A1 specifications if they chose to do it that way. It's just that they don't.

Schell: Well, yes, if a manufacturer chose to do it that way by building its

application, e.g., a PLC, on an operating system designing to meet the *TCSEC* class A1 criteria. But the reality is that if the government contract only requires the general principles, the response of the private partner is, "I want to show that what I already have meets your requirements."

Blakley: Because that's the quickest way to a purchase order, right? If you can convince somebody to buy the off-the-shelf product.

Schell: As I pointed out, David Bell reported, "NSA put all the Class A1 vendors out of business." That may be not far wrong; the ironic point is, now I hear over and over again, "We can't possibly require an operating system designed to meet class A1 because there's only one commercial vendor that can deliver that." So, the government puts all the vendors out of business and then says, "I can't have a sole source because I don't have competition." Since I was in the system acquisition business much of my career, I know that's a completely bogus answer; there are lots of components in systems for which there is only a single supplier. So, I think a first thing we can do is have the discussion as an honest discussion of substance and not just "you know you can't say 'class A1.'" I sat in a government conference where, three administrations back, we were asked to define what was needed to address cybersecurity on a national scale. Experts came together to define a whole road map for "how would you build a Manhattan project or whatever to do that?" And after having laid this out, including class A1, a major security agency representative got up and said, "We're simply not going to recommend, out of this conference, anything that says *TCSEC* because it's well known it doesn't work." He was so vociferous that other people referred to this as *TCSEC phobia*.

Blakley: In some ways, it's sort of an accurate statement: It didn't work in the market, right? And people have emotional wounds from it, and so they don't want to talk about the ways in which it did work, which were, you know, technical.

Schell: So, what are the opportunities where we can have this discussion? The president nominated and the Senate recently confirmed Chris Inglis as the first national cyber director. Chris Inglis was at the Computer Security Center when NSA was doing *TCSEC* evaluations. Chris Inglis knows what class A1 is. He's a very smart guy. It could be game-changing to have him participate in that discussion. With billions of dollars being spent on cybersecurity, let's discuss a public–private partnership like what a September 2020 LinkedIn post by Ron Ross, at NIST, described. The government owns a couple of power companies like The Tennessee Valley Authority and Bonneville Power. The government can require the partnership to deliver a prototype commercial PLC built on a class A1 operating system. By the way, Ross's November 2016 NIST SP800-160 document explicitly notes that "highly assured, kernel-based operating systems in PLCs can help achieve a high degree of system integrity and availability" for the electric grid. Someone like Chris Inglis can decide to have that discussion.

As my chief financial officer, a Harvard Law School graduate, points out, there's this notion of "raising the bar" as a way of changing a market via liability, an approach you just mentioned. The one raising the bar sets a higher standard for others to follow, in this case, to limit their liability. Once the Tennessee Valley Authority (as part of the public–private partnership) dramatically reduces the huge cybersecurity attack surface for its portion of the grid, people can no longer say "Well, this is best we can do," which is what they say today. So, there are definitely practical things we realistically can do other than blanket regulation.

The other thing I would like to address, Bob, is your "economic reasons." Earlier, you suggested we may have to "just accept that high-assurance systems are expensive." I want to challenge you a bit on that. At the time the *Orange Book* was written, those who wanted to build secure systems largely custom built their operating systems; therefore, they had to build a class A1 operating system for every type of system they wanted. No doubt, designing to meet class A1 is expensive. Literature estimates are in the range of a minimum 10–15 years and many millions of dollars. At that time, yes, it was economically infeasible for many. Fast-forward to today. The huge majority of all the IT products in the world—the billions of them—run on a handful of operating systems. For embedded devices, there is Wind River, still Microsoft CE, and various variants of Linux. Then the list gets very sparse.

Blakley: And also, the operating system isn't where the primary value add is anymore, so it's not as much of a competitive issue as it used to be.

Schell: Right, but the trick is, as shown in my September 2020 Purdue Center for Education and Research in Information Assurance and Security video on high-assurance PLCs, that the whole guts of that hard security work are in the class A1 operating system. So yes, the operating system vendor makes that investment once, but then many device manufacturers can reuse it. Within two years, we can dramatically improve our electric grid—and a lot of other things—by using a trustworthy operating system that exists. Could a manufacturer build one? No. Not in two years. But it's no longer the same economic issue: it doesn't cost a manufacturer that much more to put its device on a trustworthy operating system than it does to use an insecure one.

Blakley: So, I'm going to wrap us here, but I am going to ask our last two questions, and the first one—and actually

you were one of the people I had in mind when I formulated this question initially, before we started the whole thing—is: In the course of your career so far, what are the three or fewer things that you've learned that you really think the next generation of practitioners should remember?

Schell: Well, I have three things, which I've said:

1. First, it's scientifically impossible to build a secure cyber system without a trustworthy operating system, and by that, I mean one that's very unlikely to ever have a zero day. We didn't know that years ago. Now we know that.
2. Secondly, what wasn't true years ago, but is today, is that we have proven and mature trustworthy operating system technology that is commercially available and has been successfully deployed. In other words, we know how to use that technology to construct secure systems.
3. The third lesson is what I just discussed. Today, only a handful of core operating systems are used in the majority of the billions of information technology devices. You don't create a new operating system for individual platforms, just leverage an available trustworthy operating system.

Cranor: Great! Alright, one more question for you—we always like to end with this: What advice would you have for a young girl or boy who wants to grow up to be just like you?

Schell: Get a life! But more seriously, when I look at myself—you say *like me*—well, I'm fundamentally a leader of those individuals on my team who are pursuing some common goal, which in the military is called *the mission*. So, that's my place in life; that's not where everybody is. But if that's the case, and you're going to be like me, my advice is, when you lead, regardless of the size of the group, make your first priority the well-being of the other members of your team and enable them to be the best they can be. Be an example they are proud to follow.

Cranor: Thank you, that's great.

Blakley: It is great, and I can't tell you how much of an honor and a pleasure it has been to have you on the show, Roger. I had hoped, from the beginning, that we'd be able to do this. I just want to thank you not just for talking to us today but for everything you've done for 50 years for the information security community; it's terrific. ■

www.computer.org/cga

IEEE Computer Graphics and Applications bridges the theory and practice of computer graphics. Subscribe to *CG&A* and

- stay current on the latest tools and applications and gain invaluable practical and research knowledge,
- discover cutting-edge applications and learn more about the latest techniques, and
- benefit from *CG&A*'s active and connected editorial board.

IEEE COMPUTER SOCIETY

IEEE

Digital Object Identifier 10.1109/MSEC.2022.3171933