THE STATE OF INFOSEC EDUCATION IN ACADEMIA: PRESENT AND FUTURE DIRECTIONS

Matt Bishop
Department of Computer Science
University of California at Davis
Davis, CA 95616-8562

bishop@cs.ucdavis.edu

We are in the midst of a crisis in the deployment and use of computers, and it is getting worse every day. Our systems are not secure. They are considerably less secure than the paper systems we still use, and that are rapidly being replaced. Worse, we are not taking steps to shore up the infrastructure or systems, and we are neglecting the education necessary to base such improvements upon. Unless we devote the resources necessary to improve computer and network systems, and to educate computer scientists, operators, and users in INFOSEC, the edifice we have so painstakingly constructed will collapse, precipitating a crisis of confidence, trust, and reliance. The threat to our security as a nation is considerable; the threat to our individual privacy and identity even greater.

In this talk, I will explore the role of INFOSEC education in this crisis. I will discuss why we should care, where we are now, where we should be heading, and offer concrete suggestions about how to get there.

First, a comment about the topic. In what follows, I consider the computer security aspects of INFOSEC. INFOSEC, a portmanteau for "information security," is actually much broader; but my main interest is protecting the information on computers, and the computers themselves. The greatest threats arise in that arena, due to the marriage of technology to the information management systems and techniques.

The Importance of Computer Security

First we must ask what, exactly, is computer security? The stock definition is that "a system is secure if it conforms to a stated policy that defines what is allowed and what is disallowed" (the security policy). But this statement does not adequately convey the complexity of the problems of providing INFOSEC.

The World Wide Web provides numerous examples of how complex this issue is. Recently, the Social Security Administration made its database of earnings available over the Web. By giving a name, an address, a social security number, and the mother's maiden name, a user could access his or her past earnings as recorded by the Social Security Administration, and obtain information about their account. Was this secure? According to the Social Security Administration, it was; the data was protected by passwords (mother's maiden name). According to many others, it was not, because the passwords were easily determined. In the end, the Social Security Administration took the database off line.

Electronic mail is another example where different definitions of "security" affect the analysis. If the mail contains passwords, financial data, or expressions of love or hate, "security" means keeping the contents of the message confidential. If the mail contains information the accuracy of which is critical, such as medical data or contract information, "security" requires that the contents be unalterable while the letter is in transit; it may also require that the sender of the letter can be established to a high degree of accuracy. The flip side occurs when a sender wishes to remain anonymous, such as the student who sent a threatening letter to President Clinton. I'm sure he thought electronic mail had a serious security problem when the Secret Service showed up at his door and informed him they had traced the "anonymous" letter.

Determining how to secure a given system, in a given environment, requires analyzing the situation to determine what "security" means. From that, one can design and implement procedures, programs, and technologies to provide a level of security conforming to the needs of the system.

Some more examples will make this point clearer. A vendor designs and implements a new computer system. What role does security play? Clearly the vendor wants to provide some minimal level of security, but what is that level and how does it impact the use of the system? One school of thought is to provide mechanisms, but initially disable the mechanisms, and the managers and users can enable on those they want. Old versions of the UNIX system were distributed with this philosophy (by default, anyone could write anything). A second approach is to pick some particular policy and configure the mechanisms to enforce that policy. As an example, one vendor sells two different types of computer systems. The newer version is set to distrust other hosts on its network initially. The older version is set to trust all hosts on its network. In both cases, "the network" may well be the Internet. Now, the company at one point reconfigured the older version to distrust all hosts by default, but the outcry from small businesses was so great that the decision was reversed. The problem, it turned out, was that small businesses used local networks not connected to the Internet, and when they installed the systems, no other system on the network could talk to it. The businesses did not have the expertise to fix the problem, and so complained. This is a classic example where one sense of security ("integrity") is hampered by the need for another sense of security ("availability"). Which decision was better from the point of view of security?

Go further down to the nature of the environment in which a system was developed. One often hears that "the UNIX operating system was not designed with security in mind." That's actually not true. The UNIX system was designed in a research lab, where the only "security" necessary was to prevent one user from accidentally deleting another user's files. Given that (loosely stated) policy, the UNIX system is quite secure. But then the UNIX system moved out into less friendly environments, in which attackers could (and did, and still do) exploit flaws in programs or configuration errors to acquire privileges. In such an environment, the comment about UNIX not being designed with security in mind is quite correct.

My point is that computer security is more than mechanisms and mathematics. It includes being able to analyze a situation to figure out what constitutes security, being able to specify those requirements, being able to design a system or program to meet those requirements, being able to implement the system or program correctly, and being able to make configuration and maintenance simple.

Now, how well do we do this, in practice? As the above examples showed, rather poorly. I'll not elaborate on the Social Security fiasco, other than to say at least the designers tried to follow the above steps; their security model of the Internet, or of American citizens, was flawed. At least with paper mail, responses could be sent to a particular address (that of the person about whom the information is requested); on the Internet, this protection is impossible. With respect to electronic mail, the student who threatened President Clinton clearly could not figure out that the implementation of electronic mail failed to provide what he expected in the way of security. And the vendor who configured systems to trust all hosts on the network did not adequately analyze the assumptions made in assessing the trust between system components and the humans who would use and administer them, thereby violating the principle of psychological acceptability [4].

Nowhere do we see our failures better than in the implementation of computer systems. As I'll discuss later on, writing high-quality code is an art that all too few students ever see, and even fewer ever master. This lack shows in the systems we deploy. How many of you have ever been on a system where the screen display is replaced by a huge list of numbers, letters, and tables of dots? How many knew your gooses were cooked at that time? You're not alone. A study of utility programs on UNIX systems [3] showed that, given random input, about one-fourth to one-third of the programs dumped core. In 1 case, the kernel panicked, crashing the system! That's inexcusable, sloppy programming, and has serious consequences. In cases involving security, program crashes or incorrect behavior are not bugs; they are security holes. And most breaches of security are caused either by poor programming or by improper configuration.

Poor programming is a generic problem, because it causes security flaws under most definitions of "security." If you look at the USENET newsgroups and traffic on the security-related mailing lists, most security holes reported recently arise from buffer overflows; the attackers alter data, or change the contents of the program stack causing the program to execute machine-language routines stored on that stack (or elsewhere in memory). Checking buffer bounds is simple. True, it costs a bit more, but the cost is negligible compared to the effects of the failure to check the bounds. Also, I've never seen a study showing exactly how much overhead was added by the checking; I suspect it is considerably less than most people think. Before this, race conditions were the rage. What will be next? I wish I knew!

Improper configuration is arguably the user's, or system administrator's, fault. But why do such configuration problems arise? Configuration is often a very complex matter, and affects not only the part of the system being configured but also its interaction with the rest of the system. Most users and system administrators simply do not have the time, the experience, or the knowledge to consider all the ramifications of their configurations. So they do not configure, and trust the vendor's configuration. Or they do what seems reasonable to them. But often the vendor's configuration is for a different environment (as in the trust example above) or is counter-intuitive and without sanity checks. A perfect example of the latter is a program that manages cached DNS data. This data is critical to the correct functioning of the Internet, as it stores host name and IP address associations. Data remains in the cache for a period of time set in a configuration file; this length of time is stored as an integer number of seconds. Most folks, for whatever reason, seem to think it is a number of minutes or hours, so they put a floating-point number in the field. Say they want to purge the data after 30 minutes. If expressed as 0.5, the program reads the "0": as the integer, ignores the ".5", and sets the time-out to 0. This means that the data is never removed from the cache, which is not what the administrator intended -- and constitutes a security hazard. Now, how hard would doing a sanity check on the configuration number, and checking for a floating point number, have been? It's the "weakest link" phenomenon -- even if you configure 99.5% of your system components correctly, the remaining 0.5% usually leaves you vulnerable to attack. That these links are so weak is in part due to a failure to understand how critical simplicity and verification of configuration data is.

All this relates to INFOSEC education because it suggests a failure somewhere. Not enough computer scientists, system administrators, and programmers are learning about computer security. More knowledge and understanding of the basics of computer security, and an ability to apply these principles and techniques, would ameliorate the sorry state of security greatly. So, what can we do? How can we improve this situation? A good place to begin is with the current state of INFOSEC education.

Where We Are in INFOSEC Education

In academia, research and teaching go hand-in-hand, and it is not surprising that the four largest academic groups in INFOSEC security also have the largest concentration of students and faculty in this area. While their research overlaps somewhat, each group has carved out a general, unique niche.

The Computer Security Laboratory at the University of California at Davis has research projects in network security (including the security of the network infrastructure), testing and verification methodologies, intrusion detection, vulnerabilities analysis, policy, and auditing. The COAST Laboratory at Purdue University focuses on host security, intrusion detection, audit technologies, and computer forensics. The Center for Secure Information Systems at George Mason University conducts research in formal models, database security, and authentication technologies. The Center for Cryptography, Computer, and Network Security at the University of Wisconsin in Milwaukee focuses on the application of cryptography and cryptographic methods and their extensions. While there are other groups working on computer security in academia (at the University of Idaho, CMU, MIT, the University of Texas, the University of Maryland, Idaho State University, and Portland State University), the research programs of these four groups are the largest.

Gene Spafford presented some statistics worth repeating in his February 1997 Congressional testimony [6]. Over the last five years, these four academic institutions granted 16 Ph.D.s for security-related research. (Incidentally, 7 came from UC Davis.) Of these graduates, three went into academia. In the same time period, about 50 masters' students graduated. But these numbers, while revealing, convey only a very small part of the current state of computer security education.

Some aspects of computer security education are handled very well, some moderately well, and still others poorly. To understand the nature of the weaknesses and strengths, let's consider two different levels of INFOSEC education: graduate and undergraduate.

At the undergraduate level, teachers tend to focus more on applications of principles and operational concerns rather than the derivation and deep analysis of those principles themselves. Those are discussed, but teachers show the students how to apply the principles in very different and important situations. At this level, computer security is typically added to existing courses. For example, most books on operating systems devote a chapter or two to issues of information protection, and networking classes emphasize the need for good cryptographic protocols. Unfortunately, most of this information is presented as an adjunct to the main topic of the course and driven by that topic, so there is little unity in the material among classes. That is, the operating system class will use principles of operating systems to drive the security mechanisms and techniques discussed, and the discussion of INFOSEC security in a networking class draws upon principles of networking far more than principles of security.

This is unfortunate because students who take those classes come out with a distorted view of computer security. They do not realize that general principles guide the design of security mechanisms; that in both operating systems and networks, policy is central to the definition and implementation of computer security; and that the classes are exploring two different views of the same fundamental subject. As a result, INFOSEC security is seen as much more *ad hoc* than it is. When these students graduate and begin working in the field of computer science, they will not be able to apply the principles of security to their tasks unless the issues of security arise in the contexts of operating systems or networks. Even then, if the context is very different from that in which the issues arose in class, the students may have trouble with the security aspects of their task!

A classic example arises from network security. Network security is in large part based upon cryptography, mainly because the communications media cannot be secured; you can only protect the cryptographic keys at endpoints. One major corporation, which supplies World Wide Web browsers, understood this very well, and used the powerful RSA cipher to protect data that needed to be secured. So far, so good. But they overlooked the issue of key generation. The "unbreakable" cipher was broken in minutes by a couple of graduate students who figured out how the keys were generated, and simply began regenerating the cryptographic keys until they found ones that deciphered the messages correctly! This type of attack is rarely discussed in networking classes, yet it is a greater threat than failures in the cryptographic protocols. Are we contributing to the existing state of security by the way in which we teach it?

Superficiality seems to be common in supporting disciplines such as computer security. As another, related example, consider the discipline of programming. Everyone knows that undergraduates are taught a programming language in their first programming class. But when do they learn how to program? Programming is not simply writing code in response to an assignment, or even to a specification. Programming is crafting a program that meets the specifications, and does more -- it handles errors properly, it checks for potential problems, and basically embodies the four basic principles of robust code [2]:

1. Be paranoid about code you did not write, including library routines; expect them to fail.
2. Assume maximum stupidity; if you're writing a function, assume the caller will pass invalid parameters or bogus data.
3. Don't hand out dangerous implements; don't let the caller see what your internal data structures are, and take pains to protect them from a malicious caller.

4. Worry about cases that "can't happen;" they will, and when you least expect it.

A second course in programming should hammer these rules into the students. But my experience is that most students do not get taught these rules in such a way that they routinely apply them. This opinion is reflected in Weinberg's Second Law: "if builders built buildings the way programmers wrote programs, then the first woodpecker to come along would destroy civilization."

Undergraduates who wish to study computer security are generally relegated to graduate courses or independent study courses. Very few undergraduate computer security courses are taught now. UC Davis introduced one this year, and it was wildly successful in part because it stressed applications more than theory. (The availability of security-related jobs didn't hurt, either.) The most popular part seemed to be the lectures and assignments on writing secure code; the course evaluations showed that the brevity of this part was the major complaint!

The purpose of a graduate education is to stretch the current state of the art and the current state of knowledge. So at the graduate level, classes focus on deriving, arguing, proving, and extrapolating from fundamental principles and results and extending the underlying theory than applying it. Application is discussed when it shows interesting ramifications of the theory, or leads to interesting and novel extensions. Many of the above comments apply to these classes, the major difference being that a number of academic institutions have graduate-level classes concerned with various aspects of computer security: introductory classes, classes focusing on public policy and business, on formal methods, on databases, on cryptography, on intrusion detection, and on any particular subfield of interest to a faculty member. Classes at this level are far more flexible than undergraduate classes, and far more numerous.

Graduate education typically focuses on the design and specification of secure systems, and their development. The study of multi-level security still constitutes a major part of graduate classwork, because so much research has been done in that area; information flow, covert channels, formal models of security and integrity, and trusted computing bases all embody fundamental principles of security. However, students do not learn much about how to analyze existing systems; they may study the theory behind penetration testing, or the basic models of vulnerability analysis, but they rarely put these ideas into practice by testing an existing system, or modeling one.

Graduate education beyond the Master's level (and sometimes at the Master's level) involves more than classes, of course. It also requires research. For a master's degree, the research must contribute to the state of the art in some way, and for a doctorate, the research must extend our understanding of some aspect of security, or deepen our understanding. In other words, it must be original and contribute to the body of knowledge constituting the field of INFOSEC.

Academic computer security research is excellent in its exploration of principles. However, performing the implementations and testing, or experiments, to support the research is often a problem. Institutions suffer from inadequate or outdated equipment. Two examples should suffice. At UC Davis, we are conducting research in the network infrastructure, specifically attacks on routers, but so far all our router-related experiments have been through simulation. One router company has generously offered to let us come and use their labs, but as the company is over 2 hours away, and considerable set-up is required, this is an option we can use only occasionally. Having a router in the lab would allow us to experiment much more quickly, thereby speeding the course of the research. As another example, our vulnerabilities analysis project requires experimenting on a wide variety of computers, so we can determine how to build system-independent tools to detect potential problems. Thus far, we have only two types of systems on our network, so we cannot test or port our tools to other systems. This limits the range of our tools, and our ability to test them thoroughly enough to validate some aspects of the underlying theory.

Laboratories do not run on research alone; an infrastructure (administrative support, system administration, and so forth) is necessary. To some extent, departments try to subsidize this, but my experience is that departments do not have funding adequate for their own needs, let alone those of a growing, or mature, research laboratory. To make this personal, we recently cobbled together funds from 9 different government grants to hire an administrator (actually, a technical assistant). This amazing and dynamic individual has taken over a lot of the administrative work Karl Levitt, our postdocs, our graduate students, and I used to do. Now, I only spend 8 hours a week doing administration (report writing, not working on papers or my book; sending information to potential sponsors and current sponsors; preparing the non-technical parts of grant proposals; photocopying; scheduling meetings; updating sand installing system software; some web page designing; and so forth); the administrator has taken over the rest. With more administrative support, I could cut this time in at least half, and lift much of the burdens of system maintenance from the graduate students (we don’t have a system administrator). I do begrudge the use of that time; I understand the work is necessary, but others could do it, and probably much better than the graduate students and I could. We’d rather be teaching or researching!

Academic institutions excel at teaching principles in computer security courses. They do not teach computer security adequately as a supporting discipline in other courses, because the teachers who teach the lessons, and the authors who write the books, focus on those aspects of computer security that affect their subject.

Further, the gap between design and implementation is not covered well, even in most computer security courses.

In terms of research, the work that is done is high quality, but because of lack of necessary equipment and lack of adequate infrastructural support, the research proceeds more slowly than necessary, and is performed on equipment that is not state of the art. To emphasize this: the problem is not the theory or modeling, or the experimentation to support them; it is that the experimentation is often done via simulation rather than directly on hardware or systems with the characteristics under study. The work is good, but doing it is frustrating, and implicitly it assumes that the simulations are correct.

This very brief survey outlines the current state of INFOSEC education and research in the academic setting. To see how to improve this situation, or if improvement beyond the obvious is needed, consider what the current practice should be.

Where INFOSEC Development Should Be

The conventional wisdom is that we need to advance our understanding of modeling, security theory, and policy in order to improve the state of computer security drastically. While I agree, I think this misses one obvious point. We don't use what we know already, in either the procedural or technical arenas.

Think about it. A buffer overflow occurs when you write beyond the end of an array; this can cause the program to stop, or it may simply alter data unrelated to the buffer. We have known how to handle buffer overflows since at least the early 1960s. Compilers can generate code to check bounds. If that's too inefficient, segmented architectures provide a system-oriented mechanism for preventing overflow: make the buffer occupy a single segment. If you overflow, a segmentation trap occurs. The Burroughs systems of that time strictly delineated instructions and data; building that into the system would also prevent many of these types of attacks. This is not new technology; it's old technology. The same holds for other flaws.

In fact, we recycle flaws as if we forgot about them! In the UNIX arena, a flaw that occurred in 1993 (and would compromise a system) was the same as one found in 1983; the only difference was the name of the program involved, and how the fault was triggered. Another flaw, in an implementation of the Network File System protocol, was exactly the same as a flaw found in the 1970s in the paging of a Burroughs system. As Yogi Berra said, "it's *déjà vu* all over again!" The moral? We don't learn from our errors. We must do so!

We also do not use what we have learned. Clear statements of policies and specifications aid immeasurably in the design cycle, because they highlight the assumptions about the environment in which the programs will function. They also present the goals of the program or system clearly, and the constraints under which it must function. Even if these stated informally, the designers will know what is expected, and can design towards that goal. The goals and constraints can include security matters; for example, if the program will be writing sensitive data, confidentiality and integrity constraints should be stated explicitly. This serves two functions: to quantify the security desired somewhat, and to provide a metric for subsequent testing. But how often do specifications include this information?

Such improvements in the practice of design methodology would reduce the most pernicious, and embarrassing, part of computer security for vendors: the cycle of catch-and-patch. In this cycle, someone catches a security flaw. After considerable work, the vendor distributes a security patch. The patch typically addresses the specific flaw reported. Then the vendor learns of another security flaw. Out goes another patch. Parts of the system are becoming incrementally more secure in that single flaws are being fixed, but never is the design checked, or the flaws looked at as a whole, so other similar flaws go undiscovered. This is not cost effective -- the payment for security is incremental, and at the end, rather than up front. Worse, the patches may introduce new security holes, or aggravate the security problem (as at least one vendor discovered, to its embarrassment!)

Learning from the past, and planning designs thoroughly, will substantially improve out INFOSEC capability. There is more we can do, though.

We need to learn how to build more precise models. Currently our security-related models are crude, to say the least. We can model some aspects of systems designed for security fairly well, because the hierarchical design methods require a model from which the design is drawn. But modeling the security aspects of existing systems is a nightmare. Worse, modeling is always done at an abstract level. Details deemed irrelevant to the purpose of the model are elided or ignored. Unfortunately, in computer security, the flaws often lie in those details. The Trusted Computer System Evaluation Criteria of the Department of Defense captures this quite well; the class A1, while requiring formal proofs at the specification and design level, requires only that the "[trusted computing base] implementation must be informally shown to be consistent with the [formal top-level specification]" [1]. The next section, discussing what lies beyond A1, includes formal verification of the implementation at the software and hardware levels. Currently such verification is not practical. It needs to become practical, if not directly then through techniques such as property-based testing.

Jeremy Frank made an interesting and perceptive observation about this. The models we build often hide the problems, rather than reveal them. For example, we did some work that showed how to derive criteria for auditing from a system model, and then instantiated it using the Network File System as an example. This work skipped over the deeper question of how to create the model from which the derivation could be done. It's not intuitive, because part of the analysis is to determine what constitutes a transfer of information. Our work assumed this was known. But given a complex enough system, building a model that correctly captured all such flows could be difficult and the modelers would be likely to miss something. Perhaps techniques akin to software slicing could help here.

We need to study the formulation and implementation of policy. This includes the areas of audit analysis, configuration management, distribution of code and configuration data, and the development of modular techniques for enforcing and defining policy.

But the most important aspect of INFOSEC security is people. Programmers make mistakes. Operators make mistakes. Users make mistakes. We need to build systems that reduce the probability of human errors, and to minimize the effects of those errors. In essence, we must combine the fields of cognitive modeling, human factors, and organizational dynamics with the disciplines of software engineering and formal methods. We must understand how these errors occur, and why. Little to no work has been done in this area.

To summarize:

- We need to integrate security into all aspects of computer science education.
- We need to learn from our mistakes, and not repeat the errors from the past.
- We need to improve how we design systems and programs to account for security constraints, and we need to reduce the number of security patches necessary.
- We need to learn how to abstract models that more precisely reflect the characteristics of the system.
- We need to grasp the subtleties of policy more completely, and provide mechanisms for enforcing it with greater precision and completeness.
- We need to understand how humans interact with systems, how security problems arise from this interaction, and use this knowledge to build systems that minimize the possibility and effects of errors.

So we know where we want to go. How do we get there?

How to Improve INFOSEC Education: Meeting the Challenge

To meet these challenges, we must improve both the quality and delivery of computer security education. We need to see computer security not simply as a separate discipline, but as a multi-disciplinary science which includes elements of operating systems, networking, databases, the theory of computation, programming languages, architecture, and human/computer interaction. The body of knowledge must be incorporated as appropriate into these disciplines.

As an example, consider a second course in programming; this is typically a course in software development. We can begin to educate students in computer security at this stage, without even referring to that discipline! For example, a policy provides design constraints, so in the introductory class, we simply state the requirements of the program and the constraints under which it will function. We let the students figure out the informal specification, and require them to argue that their design meets the specification. By teaching robust programming, implementation problems such as argument checking, buffer overflows, and validation of input data become part of writing good code and are not separate aspects of writing secure programs. By spending time on the role of testing, we imbue students with the idea that systems must be validated. My point is that with a little creativity, we can ameliorate the problem of poor code in security-sensitive software.

The problems at the advanced undergraduate and graduate level are more complex. Universities and colleges provide grounding in principles, theory, the ability to analyze problems and potential solutions, and finding or predicting future problems. While industry and government are interested in these, their needs are more immediate. They want students to be educated in the systems they use. They want students who can apply technology to problems, and either solve them or figure out what new technology will solve the problems. At first glance, these roles seem contradictory. On reflection, they are complementary.

The most effective way to teach principles is to help the students discover those principles. Rather than simply stating the idea, enable the students to use systems embodying the idea they are to learn. For example, the concepts of multi-level security are simple in principle, but their use raises a host of other questions involving psychological acceptability, usability, implementation, and so forth. What better way to answer these questions than to give the students exercises on such a computer system?

This is where the marriage of industry, government, and academia can drastically improve INFOSEC education. Students learn that security cannot be provided -- indeed, defined -- in a vacuum. Real problems set parameters for applying existing theory and models, testing them, and determining their usefulness. The environment in which the problem arises sets constraints for solutions. Working with these problems teaches students to analyze not just technical issues but also non-technical issues and influences. They bring together multiple types of problems in a single situation, and show how they affect one another. They show

how money, how risk management, and how risk mitigation all influence the design and implementation of computer systems.

Consider the World Wide Web. The simplest solution to the threat of malicious downloadable executable code, such as computer viruses, is to disallow such downloads. That is not practical -- non-technical considerations suggest that, regardless of what is done, people will always down load Java applets or ActiveX programs. So, let's modify that solution -- force the browser to ask the user whether the download should proceed. In theory, great; in practice, most people will always say "yes" or call the vendor and ask how to disable that darned warning. Okay, since that won't work, how about a "sandbox", where the downloaded program is executed in a contained environment? Great idea in theory, but in practice, the construction of a universal sandbox, designed to meet all local policies is impossible. Then let's change ... but you get the idea.

Academia is changing -- slowly, but changing nonetheless. Academic institutions encourage work on problems that are presented as ill-defined or ambiguous problems; part of the challenge of the research is to make the problems well defined. And institutions are changing the standards by which they evaluate faculty; although the aphorism "publish or perish" is still true, experimental disciplines in computer science are becoming accepted as *bona fide* disciplines, even though they lead to fewer papers than more theoretical research.

Academia has a duty to educate people so that they can contribute via industry, government, or academia. Most academics, and academic institutions, take this duty very seriously -- it's an integral part of why we're in academia, and resisting the lures of more money elsewhere. We love to teach; we want our research to be useful. But we need help.

- We need more industry and government participation in selecting research topics. Nothing is more frustrating than solving a problem, only to find it is not really a problem, or the "real world" version of the problem has additional constraints that change the approach drastically. Ways to do this are through partnerships with industry in which we discuss problems and possible approaches, and work together to solve them; through internships, where members of industry come to academic institutions for a period of time to teach and work on projects with students, and where faculty and students go to industry for periods of time to work on problems of interest to the industry. One of the most common complaints of students is the lack of "real world" experience, and of industry and government is that the students lack "real world" experience. These measures would provide them.

- We need long-term funding to provide a stable base for our research. Short-term resources for tackling particular problems are helpful, but the distraction of attempting to find funds to continue our work, and to build a long-term research program, is a drain on our resources. The lack of any infrastructure support aggravates this situation; in order to hire an administrative assistant, we had to get approval from the sponsors of the 9 grants from which we drew funds. We're still short-handed. This is a complaint common to the four major labs, and trying to make up for the lack of support drains energy and time from our research.

More importantly, a stable funding base would give industry, government, and the nation a set of resources upon which they could draw without having to start from scratch. The importance of this cannot be underestimated. This base of research and knowledge can provide help and research results to deal with the crisis, and to solve the problems causing the crisis.

- We need state-of-the-art equipment. Our students learn computer science by experimentation and using systems as well as from lectures and books. The better the equipment, the better they will learn, and the less industry and government will need to train them. And the more directly applicable our research will be.

- Industry and government should fund "blue sky" research and long term, directed research. Blue sky research is speculative; it may succeed, it may fail, but the body of knowledge that comes out of it will advance the field in some manner. Remember, failure can be just as strong a result as success. Long-term research would allow us to turn our academic resources to problems that we could study thoroughly and attempt to solve in a number of different ways. Both these suggestions would produce an immeasurable amount of research and scholarship, upon which short-term projects could be built.

- Finally, industry and government should realize that demanding short-term deliverables such as software takes academia into an arena it was never meant to be in. Prototypes are built to test theories; they are in no sense production quality code, and often use designs unacceptable to production environments. Remember the software engineering adage "build the first one to throw away, and the second one to test and analyze?" A prototype is the first or second implementation. Once the theory is validated, industry should take the results and re-engineer the system to meet its specific needs, for its specific environment. Focus on the research and the results gleaned from it; we're very good at doing that. That's what we can contribute to INFOSEC research.

We have to adapt. We no longer have the luxury of fielding systems without thinking about security issues. Working together, academia, industry, and government can improve the state of INFOSEC security education and research. But the meltdown point, the point at which the computer infrastructure is about to fall apart, to become Balkanized through attacks, is almost here. We need to act, and act now.

Everyone bemoans the sorry state of computer security; so far, fewer seem willing to provide the resources to deal with the fundamental research necessary to improve the state. We should take a lesson from the Good Doctor, Dr. Seuss, who wrote a wonderful book about a youngster running away from his troubles to Solla Sollew, the land where there were no more troubles. But after a long journey, he realizes he will never get there. So he returns home [5]:

> But I've bought a big bat
> I'm all ready, you see;
> Now my troubles are going
> To have troubles with me!

May we have the wisdom to deal with our INFOSEC troubles in the same way.

Acknowledgments. Thanks to Rebecca Bace, Jeremy Frank, Karl Levitt, Vic Maconachy, and Alan Paller for helpful ideas and feedback. The contents of this note represent the opinions of the author, and not necessarily anyone else.

References

[1] Department of Defense Trusted Computer System Evaluation Criteria, DOD 5200.28-STD , sec. 4.1, p. 44 (Dec. 1985).

[2] Elliott, C., "How to Write Bomb-Proof Code," handout for COSC 23, Software Design and Implementation, Department of Mathematics and Computer Science, Dartmouth College (Jan. 1992).

[3] Miller, B., Fredriksen, L. and So, B., "An Empirical Study of the Reliability of UNIX Utilities," Communications of the ACM 33(12) pp. 33-43 (Dec. 1990).

[4] Saltzer, J. and Schroeder, M., "The Protection of Information in Computer Systems," Proceedings of the IEEE 63(9) pp. 1278-1308 (Sep. 1975).

[5] Dr. Seuss, I Had Trouble in Getting to Solla Sollew, Random House (1965).

[6] Spafford, E., Written statement submitted to the Subcommittee on Technology of the U. S. House of Representatives Committee on Science (Feb. 1997).