We Need Assurance!

Brian Snow
U. S. National Security Agency
bdsnow@nsa.gov

Abstract

When will we be secure? Nobody knows for sure – but it cannot happen before commercial security products and services possess not only enough functionality to satisfy customers' stated needs, but also sufficient assurance of quality, reliability, safety, and appropriateness for use. Such assurances are lacking in most of today's commercial security products and services. I discuss paths to better assurance in Operating Systems, Applications, and Hardware through better development environments, requirements definition, systems engineering, quality certification, and legal/regulatory constraints. I also give some examples.

1. Introduction

This is an expanded version of the "Distinguished Practitioner" address at ACSAC 2005 and therefore is less formal than most of the papers in the proceedings.

I am very grateful that ACSAC chose me as a distinguished practitioner, and I am eager to talk with you about what makes products and services secure.

Most of your previous distinguished practitioners have been from the open community; I am from a closed community, the U.S. National Security Agency, but I work with and admire many of the distinguished practitioners from prior conferences.

I spent my first 20 years in NSA doing research developing cryptographic components and secure systems. Cryptographic systems serving the U.S. government and military spanning a range from nuclear command and control to tactical radios for the battlefield to network security devices use my algorithms.

For the last 14 years, I have been a Technical Director at NSA (similar to a chief scientist or senior technical fellow in industry) serving as Technical Director for three of NSA's major mission components: the Research Directorate, the Information Assurance Directorate, and currently the Directorate for Education and Training (NSA's Corporate University). Throughout these years, my mantra has been, "Managers are responsible for doing things right; Technical Directors are responsible for finding the right things to do."

There are many things to which NSA pays attention in developing secure products for our National Security Customers to which developers of commercial security offerings also need to pay attention, and that is what I want to discuss with you today.

2. Setting the context

The RSA Conference of 1999 opened with a choir singing a song whose message is still valid today: "Still Haven't Found What I'm Looking For". The reprise phrase was . . . *"When will I be secure? Nobody knows for sure. But I still haven't found what I'm looking for!"*

That sense of general malaise still lingers in the security industry; why is that? Security products and services should stop malice in the environment from damaging their users. Nevertheless, too often they fail in this task. I think it is for two major reasons.

First, too many of these products are still designed and developed using methodologies assuming random failure as the model of the deployment environment rather than assuming malice. There is a world of difference!

Second, users often fail to characterize the nature of the threat they need to counter. Are they subject only to a generic threat of an opponent seeking some weak system to beat on, not necessarily theirs, or are they subject to a targeted attack, where the opponent wants something specific of theirs and is willing to focus his resources on getting it?

The following two simple examples might clarify this.

Example 1: As a generic threat, consider a burglar roaming the neighborhood wanting to steal a VCR. First, understand his algorithm: Find empty house

(dark, no lights) try door; if open, enter, if VCR – take. If the door is resistant, or no VCR is present, find another dark house.

Will the burglar succeed? Yes, he will probably get a VCR in the neighborhood. Will he get yours? What does it take to stop him? Leave your lights on when you go out (9 cents a kilowatt-hour) and lock your door. That is probably good enough to stop the typical generic burglar.

Example 2: As a targeted threat, assume you have a painting by Picasso worth $250,000 hanging above your fireplace, and an Art thief knows you have it and he wants it. What is his algorithm? He watches your house until he sees the whole family leave. He does not care if the lights are on or not. He approaches the house and tries the door; if open, he enters. If locked, he kicks it in. If the door resists, he goes to a window. If no electronic tape, he breaks the glass and enters. If electronic tape is present, he goes to the siding on the house, rips some off, then tears out the fiberboard backing, removes the fiberglass insulation, breaks though the interior gypsum board, steps between the studs, and finally takes the painting and leaves.

It takes more effort to counter a targeted threat. In this case, typically a burglar alarm system with active polling and interior motion sensors as a minimum (brick construction would not hurt either). With luck, this should be enough to deter him. If not, at least there should be increased odds of recovery due to hot pursuit once the alarms go off.

There is no such thing as perfect security; you need to know how much is enough to counter the threat you face, and this changes over time.

3. What do we need?

NSA has a proud tradition during the past 53 years of providing cryptographic hardware, embedded systems, and other security products to our customers. Up to a few years ago, we were a sole-source provider. In recent years, there has come to be a commercial security industry that is attractive to our customers, and we are in an unaccustomed position of having to "compete." There is nothing wrong with that. *If* industry can meet our customer's needs, so be it.

Policy and regulation still require many of our customers to accept Government advice on security products. However, they really press us to recommend commercial solutions for cost savings and other reasons. Where we can, we do so. However, we do not do it very often because we still have not found what *we* are looking for – assurance.

Assurance is essential to security products, but it is missing in most commercial offerings today. The major shortfall is absence of assurance (or safety) mechanisms in *software*. If my car crashed as often as my computer does, I would be dead by now.

In fact, compare the software industry to the automobile industry at two points in its history, the 1930s and today. In 1930, the auto industry produced cars that could go 60 mph or faster, looked nice, and would get you from here to there. Cars "performed" well, but did not have many "safety features." If you were in an accident at high-speed, you would likely die.

The car industry today provides air bags, seat belts, crush zones, traction control, anti-skid braking, and a host of other safety details (many required by legislation) largely invisible to the purchaser. Do you *regularly* use your seat belt? If so, you realize that users *can* be trained to want and to use assurance technology!

The software security industry today is at about the same stage as the auto industry was in 1930; it provides performance, but offers little safety. For both cars and software, the issue is really assurance.

Yet what we need in security products for high-grade systems in DoD is more akin to a military tank than to a modern car! Because the environment in which our products must survive and function (battlefields, etc.) has malice galore.

I am looking forward to, and need, convergence of government and commercial security products in two areas: assurance, and common standards. Common standards will come naturally, but assurance will be harder – so I am here today as an evangelist for assurance techniques.

Many vendors tell me that users are not willing to pay for assurance in commercial security products; I would remind you that Toyota and Honda penetrated U.S. Markets in the 70's by differentiating themselves from other brands by improving reliability and quality! What software vendor today will become the "Toyota" of this industry by selling robust software?

4. Assurance: first definition

What do I mean by assurance? I'll give a more precise definition later, but for now it suffices to say that assurance work makes a user (or accreditor) more confident that the system works as intended, without flaws or surprises, even in the presence of malice.

We analyze the system at design time for potential problems that we then correct. We test prototype devices to see how well they perform under stress or when used in ways beyond the normal specification. Security acceptance testing not only exercises the product for its expected behavior given the expected

environment and input sequences, but also tests the product with swings in the environment outside the specified bounds and with improper inputs that do not match the interface specification. We also test with proper inputs, but in an improper sequence. We anticipate malicious behavior and design to counter it, and then test the countermeasures for effectiveness. We expect the product to behave safely, even if not properly, under any of these stresses. If it does not, we redesign it.

I want functions *and* assurances in a security device. We do not "beta-test" on the customer; if my product fails, someone might die.

Functions are typically visible to the user and commanded through an interface. Assurances tend to be invisible to the user but keep him safe anyway.

Examples would be thicker insulation on a power wire to reduce the risk of shock, and failure analysis to show that no single transistor failure will result in a security compromise.

Having seat belts in a car provides a safety function. Having them made of nylon instead of cotton is the result of assurance studies that show nylon lasts longer and retains its strength better in the harsh environment of a car's interior.

Assurance is best addressed during the initial design and engineering of security systems – not as after-market patches. The earlier you include a security architect or maven in your design process, the greater is the likelihood of a successful and robust design. The usual quip is, "He who gets to the interface first, wins".

When asked to predict the state of "security ten years from now," I focus on the likely absence of assurance, rather than the existence of new and wonderful things.

Ten years from now, there will still be security-enhanced software applications vulnerable to buffer overflow problems. These products will not be secure, but will be sold as such.

Ten years from now, there will still be security-enhanced operating systems that will crash when applications misbehave. They will not be secure either.

Ten years from now, we will have sufficient functionality, plenty of performance, but not enough assurance.

Otherwise, predicting ten years out is simply too hard in this industry, so I will limit myself to about five years. Throughout the coming five-year span, I see little improvement in assurance, hence little true security offered by the industry.

5. The current state of play

Am I depressed about this state of affairs? Yes, I am. The scene I see is products and services sufficiently robust to counter many (but not all) of the "hacker" attacks we hear so much about today, but not adequate against the more serious but real attacks mounted by economic enemies, organized crime, nation states, and yes, terrorists.

We will be in a truly dangerous stance: we will think we are secure (and act accordingly) when in fact we are not secure.

The serious enemy knows how to hide his activities. What is the difference between a hacker and a more serious threat such as organized crime? The hacker wants a *score*, and bragging rights for what he has obviously defaced or entered. Organized crime wants a *source*, is willing to work long, hard, and quietly to get in, and once in, wants to stay invisible and continue over time to extract what it needs from your system.

Clearly, we need confidence in security products; I hope we do not need a major bank-failure or other disaster as a wake-up call before we act.

The low-level hackers and "script-kiddies" who are breaking systems today and are either bragging about it or are dumb enough to be caught, are providing some of the best advertising we could ask for to justify the need for assurance in security products.

They demonstrate that assurance techniques (*barely*) adequate for a benign environment simply will not hold up in a malicious environment, so we *must* design to defeat malice. Believe me – there is malice out there, beyond what the "script-kiddies" can mount.

However, I do fear for the day when the easy threats are countered – that we may then stop at that level, rather than press on to counter the serious and pernicious threats that can stay hidden.

During the next several years, we need major pushes and advances in three areas: Scalability, Interoperability, and Assurance. I believe that market pressures will provide the first two, but not the last one – assurance.

There may or may not be major breakthroughs in new security functions; but we really do not need many new functions or primitives – if they come, that is nice. If they do not, we can make do with what we have.

What we really need but are not likely to get is greater levels of assurance. That is sad, because despite the real need for additional research in assurance technology, the real crime is that we fail to

use fully that which we already have in hand! We need to better use those confidence-improving techniques that we do have, and continue research and development efforts to refine them and find others.

I am not asking for the development of new science; the safety and reliability communities (and others) know how to do this – go and learn from them.

You are developers and marketers of security products, and I am sorry that even as your friend I must say, "Shame on you. You should build them better!" It is a core quality-of-implementation issue. The fact that teen-age hackers can penetrate many of your devices from home is an abysmal statement about the security-robustness of the products.

6. Assurance: second definition

It is time for a more precise definition. Assurances are confidence-building activities demonstrating that

1. The system's security policy is internally consistent and reflects the requirements of the organization,
2. There are sufficient security functions to support the security policy,
3. The system functions meet a desired set of properties and *only* those properties,
4. The functions are implemented correctly, and
5. The assurances *hold up* through the manufacturing, delivery, and life cycle of the system.

We provide assurance through structured design processes, documentation, and testing, with greater assurance provided by more processes, documentation, and testing.

I grant that this leads to increased cost and delayed time-to-market – a severe one-two punch in *today's* marketplace; but your customers are growing resistive and are beginning to expect, and to demand, better products *tomorrow*. They are near the point of chanting, "I'm mad as hell, and I'm not going to take it anymore!"

Several examples of assurance techniques come to mind; I will briefly discuss some in each of the following six areas: operating systems, software modules, hardware features, systems engineering, third party testing, and legal constraints.

7. Operating systems

Even if operating systems are not truly secure, they can at least remain benign (not actively malicious) if they would simply enforce a digital signature check on every critical module prior to each execution. Years ago, NSA's research organization wrote test code for a UNIX system that did exactly that. The performance degraded about three percent. This is something that is doable!

Operating Systems should be self-protective and enforce (at a minimum) separation, least-privilege, process-isolation, and type-enforcement.

They should be aware of and enforce security policies! Policies drive requirements. Recall that Robert Morris, a prior chief scientist for the National Computer Security Center, once said: "Systems built without requirements cannot fail; they merely offer surprises – usually unpleasant!"

Given today's common hardware and software architectural paradigms, operating systems security is a major primitive for secure systems – you will not succeed without it. This area is so important that it needs all the emphasis it can get. It is the current "black hole" of security.

The problem is innately difficult because from the beginning (ENIAC, 1944), due to the high cost of components, computers were built to share resources (memory, processors, buses, etc.). If you look for a one-word synopsis of computer design philosophy, it was and is SHARING. In the security realm, the one word synopsis is SEPARATION: keeping the bad guys away from the good guys' stuff!

So today, making a computer secure requires imposing a "separation paradigm" on top of an architecture built to share. That is tough! Even when partially successful, the residual problem is going to be covert channels. We really need to focus on making a secure computer, not on making a computer secure – the point of view changes your beginning assumptions and requirements!

8. Software modules

Software modules should be well documented, written in certified development environments, (ISO 9000, SEI-CMM level five, Watts Humphrey's Team Software Process and Personal Software Process (TSP/PSP), etc.), and *fully* stress-tested at their interfaces for boundary-condition behavior, invalid inputs, and proper commands in improper sequences.

In addition to the usual quality control concerns, *bounds checking* and *input scrubbing* require special attention. For bounds checking, verify that inputs are of the expected type: if numeric, in the expected range; if character strings, the length does not exceed the internal buffer size. For input scrubbing, implement reasonableness tests: if an input should be a single word of text, a character string containing multiple words is wrong, even if it fits in the buffer.

A strong quality control regime with aggressive bounds checking and input scrubbing will knock out the vast majority of today's security flaws.

We also need good configuration control processes and design modularity.

A good security design process requires review teams as well as design teams, and no designer should serve on the review team. They cannot be critical enough of their own work. Also in this world of multi-national firms with employees from around the world, it may make sense to take the national affinity of employees into account, and not populate design and review teams for a given product with employees of the SAME nationality or affinity. Half in jest I would say that if you have Israelis on the design team put Palestinians on the review team; or if Germans are on one, put French on the other. . . .

Use formal methods or other techniques to assure modules meet their specifications exactly, with no extraneous or unexpected behaviors – especially embedded malicious behavior.

Formal methods have improved dramatically over the years, and have demonstrated their ability to reduce errors, save time, and even save dollars! This is an under-exploited and very promising area deserving more attention.

I cite two examples of formal methods successes: The Microsoft SLAM static driver verifier effort coming on line in 2005, and Catherine Meadows' NRL Protocol Analyzer detecting flaws in the IKE (Internet Key Exchange) protocol in 1999. You may have your own recent favorites.

As our systems become more and more complex, the need for, and value of, formal methods will become more and more apparent.

9. Hardware features

Consider the use of smartcards, smart badges, or other hardware tokens for especially critical functions. Although more costly than software, when properly implemented the assurance gain is great. The form-factor is not as important as the existence of an isolated processor and address space for assured operations – an "Island of Security," if you will. Such devices can communicate with each other through secure protocols and provide a web of security connecting secure nodes located across a sea of insecurity in the global net.

I find it depressing that the hardware industry has provided hardware security functionality (from the Trusted Platform Group and others) now installed in processors and motherboards that is not yet accessed or used by the controlling software, whether an OS or an application.

10. Security systems engineering

How do we get high assurance in commercial gear?

a) How can we trust, or

b) If we cannot trust, how can we safely use, security gear of unknown quality?

Note the difference in the two characterizations above: *how we phrase the question may be important*. For my money, I think we need more focus on how to use safely security gear of unknown quality (or of uncertain provenance).

I do not have a complete answer on how to handle components of unknown quality, but my thoughts lean toward systems engineering approaches somewhat akin to what the banking industry does in their systems. No single component, module, or person knows enough about the overall transaction processing system to be able to mount a successful attack at any one given access point. To be successful the enemy must have access at multiple points and a great deal of system architecture data.

Partition the system into modules with "blinded interfaces" and limited authority where the data at any one interface are insufficient to develop a complete attack. Further, design cooperating modules to be "mutually suspicious," auditing and alarming each other's improper behavior to the extent possible.

For example: if you are computing interest to post to accounts there is no need to send the complete account record to a subroutine to adjust the account balance. Just send the current balance and interest rate, and on return store the result in the account record. Now the interest calculating subroutine *cannot* see the data on the account owner, and therefore cannot target specific accounts for theft or other malicious action. We need to trust the master exec routine, but minimize the number of subroutines we need to trust. Yes, I know this is over-simplified, but you get my drift.

In addition, to guard against "unintended extra functionality" within given hardware modules or software routines, the development philosophy needs to enforce something akin to "no-lone zones" in that no single designer or coder can present a "black-box" (or proprietary?) effort to the system design team that is tested only at its interfaces and is then accepted.

Review all schematics and code (in detail, line by line) for quality and "responsive to stated requirement" goals. This review should be by parties independent of the designer. This is expensive, but not

far from processes required today in many quality software development environments to address reliability and safety concerns.

This of course requires all tools (compilers, CAD support, etc.) used in the development environment to be free of malice; that can be a major hurdle and a difficult assurance task in and of itself (remember the Thompson compiler in "Reflections on Trusting Trust, CACM 1983)!

The "Open Source" movement may also provide value in this area. There are pluses and minuses with open source, but from the security viewpoint, I believe it is primarily a plu**s.**

Further architectural constraints may be imposed to make up for deficiencies in certain modules. Rather than (or in addition to) encryption in application processes prior to transmission to other sites which could be bypassed or countered by a malicious operating system, you might require site-to-site transmissions to go through an encrypting modem or other in-line, non-bypassable link encryptors.

Link encryption in addition to application layer encryption is an example of a "Defense in Depth" strategy that attempts to combine several weak or possibly flawed mechanisms in a fashion robust enough to provide protection at least somewhat stronger than the strongest component present.

Synergy, where the strength of the whole is greater than the sum of the strength of the parts, is highly desirable but not likely. We must avoid at all costs the all-too-common result where the system strength is less than the strength offered by the strongest component, and in some worst cases less than the weakest component present. Security is so very fragile under composition; in fact, secure composition of components is a major research area today.

Good *system* security design today is an art, not a science. Nevertheless, there are good practitioners out there that can do it. For instance, some of your prior distinguished practitioners fit the bill.

This area of "safe use of inadequate components" is one of our hardest problems, but an area where I expect some of the greatest payoffs in the future and where I invite you to spend effort.

11. Third party testing

NIST (and NSA) provide third-party testing in the National Information Assurance Partnership Laboratories (NIAP labs), but Government certification programs will only be successful if users see the need for something other than vendor claims of adequacy or what I call "proof by emphatic assertion – Buy me, I'm Good."

If not via NIST or other government mechanism, then the industry must provide *third-party* mediation for vendor security claims via consortia or other mechanisms to provide *independent* verification of vendor claims *in a way understandable by users*.

12. Market/legal/regulatory constraints

Market pressures are changing, and may now help drive more robust security functionality. The emergence of e-commerce in the past decade as a driver for secure internet financial transactions is certainly helpful, as is the entertainment industry's focus on digital rights management. These industries certainly want security laid on correctly and robustly!

I hope citizens will be able to use the emerging mechanisms to protect personal data in their homes, as well as industry using the mechanisms to protect industry's fiscal and intellectual property rights. It is simply a matter of getting the security architecture right.

I wonder if any of the industry consortia working on security for digital rights management and/or electronic fiscal transactions have citizen advocates sitting on their working groups.

Lawsuits might help lead to legal "fitness-for-use" criteria for software products – much as other industries face today. This could be a big boon to assurance – liability for something other than the quality of the media on which a product is delivered!

Recall that failure to deliver expected functionality can be viewed, in legal parlance, as providing an "attractive nuisance" and is often legally actionable.

One example is a back yard swimming pool with no fence around it. If a neighbor's child drowns in it, you can be in deep trouble for providing an attractive nuisance. Likewise, if you do a less than adequate job of shoveling snow from your walk in winter (providing the appearance of usability) you can be liable if someone slips on the ice you left on the surface. Many software security products today are attractive nuisances!

All you need do is to Google "Software Quality Lawsuits" or a similar phrase, and you can find plenty of current examples of redress sought under law for lack of quality in critical software. Do not attempt to manage defects in software used in life-critical applications. Remove them during the development and testing processes! People have died due to poor software in medical devices, and the courts are now engaged; the punitive awards can be significant.

One example of a lawsuit already settled: *General Motors Corp. v. Johnston* (1992). A truck stalled and was involved in an accident because of a defect in a PROM, leading to the death of a seven-year old child. An award of $7.5 million in punitive damages against GM followed, in part due to GM knowing of the fault, but doing nothing.

There are social processes outside the courts that can also drive vendors toward compliance with quality standards.

One of the most promising recent occurrences in the insurance industry was stated in the report of Rueschlikon 2005 (a conference serving the insurance industry). Many participants felt that, "The insurance industry's mechanisms of premiums, deductibles, and eligibility for coverage can incent best practices and create a market for security . . . This falls in line with the historic role played by the insurance industry to create incentives for good practices, from healthcare to auto safety . . . Moreover, the adherence to a set of best practices suggest that if they were not followed, firms could be held liable for negligence."

Bluntly, if your security product lacks sufficient robustness in the presence of malice, your customers will have to pay more in insurance costs to mitigate their risks.

How the insurance industry will measure best practices and measure compliance are still to be worked out, but I believe *differential* pricing of business disaster recovery insurance based in part on quality/assurance (especially of security components) is a great stride forward in bringing market pressure to bear in this area!

13. Summary

In closing, I reiterate that what we need most in the future is more assurance rather than more functions or features. The malicious environment in which security systems must function *absolutely requires* the use of strong assurance techniques.

Remember: most attacks today result from failures of assurance, not failures of function.

Rather than offer predictions, try for a self-fulfilling prophecy – each of us should leave this conference with a stronger commitment to using available assurance technology in products! It is not adequate to *have* the techniques; we must *use* them!

We have our work cut out for us; let's go do it.

In closing, I would like to thank Steven Greenwald, Brad Martin, and Greg Shipley for their insights and help in preparing this article.