The Six Dumbest Ideas in Computer Security

There's lots of innovation going on in security - we're inundated with a steady stream of new stuff and it all sounds like it works just great. Every couple of months I'm invited to a new computer security conference, or I'm asked to write a foreword for a new computer security book. And, thanks to the fact that it's a topic of public concern and a "safe issue" for politicians, we can expect a flood of computer security-related legislation from lawmakers. So: computer security is definitely still a "hot topic." But why are we spending all this time and money and still having problems?

Let me introduce you to the ***six dumbest ideas in computer security***. What are they? They're the ***anti***-good ideas. They're the braindamage that makes your $100,000 ASIC-based turbo-stateful packet-mulching firewall transparent to hackers. Where do ***anti***-good ideas come from? They come from misguided attempts to do the impossible - which is another way of saying "trying to ignore reality." Frequently those misguided attempts are sincere efforts by well-meaning people or companies who just don't fully understand the situation, but other times it's just a bunch of savvy entrepreneurs with a well-marketed piece of junk they're selling to make a fast buck. In either case, these dumb ideas are the fundamental reason(s) why all that money you spend on information security is going to be wasted, unless you somehow manage to avoid them.

For your convenience, I've listed the dumb ideas in descending order from the most-frequently-seen. If you can avoid falling into the the trap of the first three, you're among the few true computer security elite.

#1) Default Permit

This dumb idea crops up in a lot of different forms; it's incredibly persistent and difficult to eradicate. Why? Because it's so attractive. Systems based on "Default Permit" are the computer security equivalent of empty calories: tasty, yet fattening.

The most recognizable form in which the "Default Permit" dumb idea manifests itself is in firewall rules. Back in the very early days of computer security, network managers would set up an internet connection and decide to secure it by turning off incoming telnet, incoming rlogin, and incoming FTP. Everything else was allowed through, hence the name "Default Permit." This put the security practitioner in an endless arms-race with the hackers. Suppose a new vulnerability is found in a service that is not blocked - now the administrators need to decide whether to deny it or not, hopefully, before they got hacked. A lot of organizations adopted "Default Permit" in the early 1990's and convinced themselves it was OK because "hackers will never bother to come after us." The 1990's, with the advent of worms, should have killed off "Default Permit" forever but it didn't. In fact, most networks today are still built around the notion of an open core with no segmentation. That's "Default Permit."

Another place where "Default Permit" crops up is in how we typically approach code execution on our systems. The default is to permit anything on your machine to execute if you click on it, unless its execution is denied by something like an antivirus program or a spyware blocker. If you think about that for a few seconds, you'll realize what a dumb idea that is. On my computer here I run about 15 different applications on a regular basis. There are probably another 20 or 30 installed that I use every couple of months or so. I still don't understand why operating systems are so dumb that they let any old virus or piece of spyware execute without even asking me. That's "Default Permit."

A few years ago I worked on analyzing a website's security posture as part of an E-banking security project. The website had a load-balancer in front of it, that was capable of re-vectoring traffic by URL, and my client wanted to use the load-balancer to deflect worms and hackers by re-vectoring attacks to a black hole address. Re-vectoring attacks would have meant adopting a policy of "Default Permit" (i.e.: if it's not a known attack, let it through) but instead I talked them into adopting the opposite approach. The load-balancer was configured to re-vector any traffic ***not*** matching a complete list of correctly-structured URLs to a server that serves up image data and 404 pages, which is running a special locked-down configuration. Not surprisingly, that site has withstood the test of time quite well.

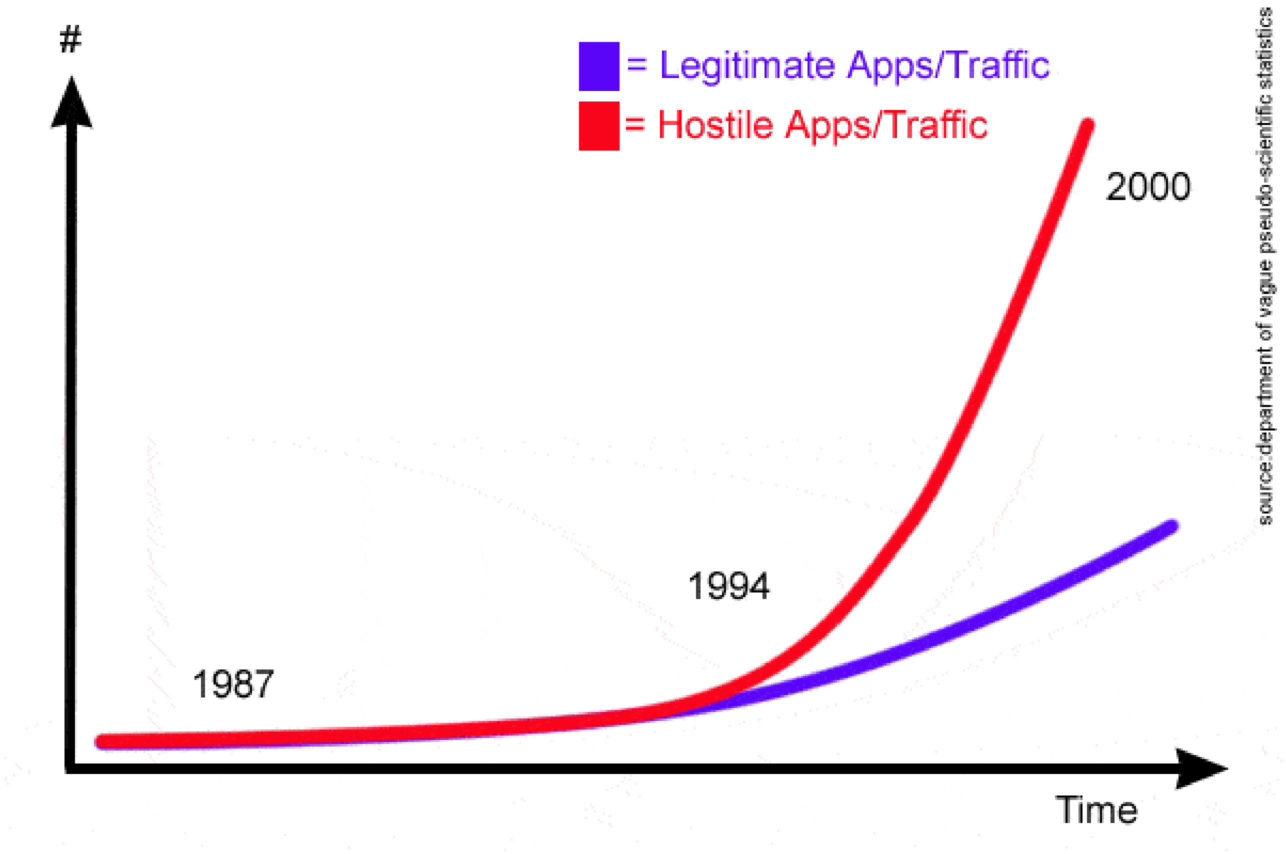
One clear symptom that you've got a case of "Default Permit" is when you find yourself in an arms race with the hackers. It means that you've put yourself in a situation where what you don't know ***can*** hurt you, and you'll be doomed to playing keep ahead/catch-up.

The opposite of "Default Permit" is "Default Deny" and it is a ***really*** good idea. It takes dedication, thought, and understanding to implement a "Default Deny" policy, which is why it is so seldom done. It's not that much harder to do than "Default Permit" but you'll sleep much better at night.

#2) Enumerating Badness

Back in the early days of computer security, there were only a relatively small number of well-known security holes. That had a lot to do with the widespread adoption of "Default Permit" because, when there were only 15 well-known ways to hack into a network, it was possible to individually examine and think about those 15 attack vectors and block them. So security practitioners got into the habit of "Enumerating Badness" - listing all the bad things that we know about. Once you list all the badness, then you can put things in place to detect it, or block it.

Figure 1: The "Badness Gap"



Why is "Enumerating Badness" a dumb idea? It's a dumb idea because sometime around 1992 the amount of Badness in the Internet began to vastly outweigh the amount of Goodness. For every harmless, legitimate, application, there are dozens or hundreds of pieces of malware, worm tests, exploits, or viral code. Examine a typical antivirus package and you'll see it knows about 75,000+ viruses that might infect your machine. Compare that to the legitimate 30 or so apps that I've installed on my machine, and you can see it's rather dumb to try to track 75,000 pieces of Badness when even a simpleton could track 30 pieces of Goodness. In fact, if I were to simply track the 30 pieces of Goodness on my machine, and allow nothing else to run, I would have simultaneously solved the following problems:

Spyware
Viruses
Remote Control Trojans
Exploits that involve executing pre-installed code that you don't use regularly

Thanks to all the marketing hype around disclosing and announcing vulnerabilities, there are (according to some industry analysts) between 200 and 700 new pieces of Badness hitting the Internet every month. Not only is "Enumerating Badness" a dumb idea, it's gotten dumber during the few minutes of your time you've bequeathed me by reading this article.

Now, your typical IT executive, when I discuss this concept with him or her, will stand up and say something like, "That sounds great, but our enterprise network is *really* complicated. Knowing about all the different apps that we rely on would be impossible! What you're saying sounds reasonable until you think about it and realize how absurd it is!" To which I respond, "How can you call yourself a 'Chief Technology Officer' if you have no idea what your technology is doing?" A CTO isn't going to know detail about every application on the network, but if you haven't got a vague idea what's going on it's impossible to do capacity planning, disaster planning, security planning, or virtually any of the things in a CTO's charter.

In 1994 I wrote a firewall product that needed some system log analysis routines that would alert the administrator in case some kind of unexpected condition was detected. The first version used "Enumerating Badness" (I've been dumb, too) but the second version used what I termed "Artificial Ignorance" - a process whereby you *throw away the log entries you know aren't interesting*. If there's anything left after you've thrown away the stuff you know isn't interesting, then the leftovers *must* be interesting. This approach worked amazingly well, and detected a number of very interesting operational conditions and errors that it simply never would have occurred to me to look for.

"Enumerating Badness" is the idea behind a huge number of security products and systems, from anti-virus to intrusion detection, intrusion prevention, application security, and "deep packet inspection" firewalls. What these programs and devices do is *outsource* your process of knowing what's good. Instead of you taking the time to list the 30 or so legitimate things you need to do, it's easier to pay $29.95/year to someone else who will try to maintain an exhaustive list of all the evil in the world. Except, unfortunately, your badness expert will get $29.95/year for the antivirus list, another $29.95/year for the spyware list, and you'll buy a $19.95 "personal firewall" that has application control for network applications. By the time you're done paying other people to enumerate all the malware your system could come in contact with, you'll more than double the cost of your "inexpensive" desktop operating system.

One clear symptom that you have a case of "Enumerating Badness" is that you've got a system or software that needs signature updates on a regular basis, or a system that lets past a new worm that it hasn't seen before. The cure for "Enumerating Badness" is, of course, "Enumerating Goodness." Amazingly, there is virtually no support in operating systems for such software-level controls. I've tried using Windows XP Pro's Program Execution Control but it's oriented toward "Enumerating Badness" and is, itself a dumb implementation of a dumb idea.

In a sense, "Enumerating Badness" is a special dumb-case of "Default Permit" - our #1 dumb computer security idea. But it's so prevalent that it's in a class by itself.

#3) Penetrate and Patch

There's an old saying, "You cannot make a silk purse out of a sow's ear." It's pretty much true, unless you wind up using so much silk to patch the sow's ear that eventually the sow's ear is completely replaced with silk. Unfortunately, when buggy software is fixed it is almost always fixed through the addition of new code, rather than the removal of old bits of sow's ear.

"Penetrate and Patch" is a dumb idea best expressed in the BASIC programming language:

```
10 GOSUB LOOK_FOR_HOLES
20 IF HOLE_FOUND = FALSE THEN GOTO 50
30 GOSUB FIX_HOLE
40 GOTO 10
50 GOSUB CONGRATULATE_SELF
60 GOSUB GET_HACKED_EVENTUALLY_ANYWAY
70 GOTO 10
```

In other words, you attack your firewall/software/website/whatever from the outside, identify a flaw in it, fix the flaw, and then go back to looking. One of my programmer buddies refers to this process as "turd polishing" because, as he says, it doesn't make your code any less smelly in the long run but management might enjoy its improved, shiny, appearance in the short term. In other words, the problem with "Penetrate and Patch" is not that it makes your code/implementation/system *better by design*, rather it merely makes it *toughened by trial and error*. Richard Feynman's "Personal Observations on the Reliability of the Space Shuttle" used to be required reading for the software engineers that I hired. It contains some profound thoughts on expectation of reliability and how it is achieved in complex systems. In a nutshell its meaning to programmers is: "Unless your system was *supposed to be hackable* then it shouldn't be hackable."

"Penetrate and Patch" crops up all over the place, and is the primary dumb idea behind the current fad (which has been going on for about 10 years) of vulnerability disclosure and patch updates. The premise of the "vulnerability researchers" is that they are helping the community by finding holes in software and getting them fixed before the hackers find them and exploit them. The premise of the vendors is that they are doing the right thing by pushing out patches to fix the bugs before the hackers and worm-writers can act upon them. Both parties, in this scenario, are being dumb because if the vendors were writing code that had been designed to be secure and reliable then vulnerability discovery would be a tedious and unrewarding game, indeed!

Let me put it to you in different terms: ***if "Penetrate and Patch" was effective, we would have run out of security bugs in Internet Explorer by now***. What has it been? 2 or 3 a month for 10 years? If you look at major internet applications you'll find that there are a number that consistently have problems with security vulnerabilities. There are also a handful, like PostFix, Qmail, etc, that were engineered to be compartmented against themselves, with modularized permissions and processing, and - not surprisingly - they have histories of amazingly few bugs. The same logic applies to "penetration testing." There are networks that I know of which have been "penetration tested" any number of times and are continually getting hacked to pieces. That's because their design (or their security practices) are so fundamentally flawed that no amount of turd polish is going to keep the hackers out. It just keeps managers and auditors off of the network administrator's backs. I know other networks that it is, literally, pointless to "penetration test" because they were designed from the ground up to be permeable only in certain directions and only to certain traffic destined to carefully configured servers running carefully secured software. Running a "penetration test" for Apache bugs is completely pointless against a server that is running a custom piece of C code that is running in a locked-down portion of an embedded system. So, "Penetrate and Patch" is pointless either because you know you're going to find an endless litany of bugs, or because you know you're not going to find anything comprehensible. Pointless is dumb.

One clear symptom that you've got a case of "Penetrate and Patch " is when you find that your system is always vulnerable to the "bug of the week." It means that you've put yourself in a situation where every time the hackers invent a new weapon, it works against you. Doesn't that sound dumb? Your software and systems should be *secure by design* and should have been *designed with flaw-handling in mind*.

#4) Hacking is Cool

One of the best ways to get rid of cockroaches in your kitchen is to scatter bread-crumbs under the stove, right? Wrong! That's a dumb idea. One of the best ways to discourage hacking on the Internet is to give the hackers stock options, buy the books they write about their exploits, take classes on "extreme hacking kung fu" and pay them tens of thousands of dollars to do "penetration tests" against your systems, right? Wrong! "Hacking is Cool" is a really dumb idea.

Around the time I was learning to walk, Donn Parker was researching the behavioral aspects of hacking and computer security. He says it better than I ever could: *"Remote computing freed criminals from the historic requirement of proximity to their crimes. Anonymity and freedom from personal victim confrontation increased the emotional ease of crime, i.e., the victim was only an inanimate computer, not a real person or enterprise. Timid people could become criminals. The proliferation of identical systems and means of use and the automation of business made possible and improved the economics of automating crimes and constructing powerful criminal tools and scripts with great leverage."*

Hidden in Parker's observation is the awareness that ***hacking is a social problem***. It's not a technology problem, at all. "*Timid people could become criminals.*" The Internet has given a whole new form of elbow-room to the badly socialized borderline personality. The #4th dumbest thing information security practitioners can do is implicitly encourage hackers by lionizing them. The media plays directly into this, by portraying hackers, variously, as "whiz kids" and "brilliant technologists" - of course if you're a reporter for CNN, anyone who can install Linux probably *does* qualify as a "brilliant technologist" to you. I find it interesting to compare societal reactions to hackers as "whiz kids" versus spammers as "sleazy con artists." I'm actually heartened to see that the spammers, phishers, and other scammers are adopting the hackers and the techniques of the hackers - this will do more to reverse society's view of hacking than any other thing we could do.

If you're a security practitioner, teaching yourself how to hack is also part of the "Hacking is Cool" dumb idea. Think about it for a couple of minutes: teaching yourself a bunch of exploits and how to use them means you're investing your time in learning a bunch of tools and techniques that are going to go stale as soon as everyone has patched that particular hole. It means you've made part of your professional skill-set dependent on "Penetrate and Patch" and you're going to have to be part of the arms-race if you want that skill-set to remain relevant and up-to-date. Wouldn't it be more sensible to learn how to design security systems that are hack-proof than to learn how to identify security systems that are dumb?

My prediction is that the "Hacking is Cool" dumb idea will be a dead idea in the next 10 years. I'd like to fantasize that it will be replaced with its opposite idea, "Good Engineering is Cool" but so far there is no sign that's likely to happen.

#5) Educating Users

"Penetrate and Patch" can be applied to human beings, as well as software, in the form of user education. On the surface of things, the idea of "Educating Users" seems less than dumb: education is always good. On the other hand, like "Penetrate and Patch" ***if it was going to work, it would have worked by now***. There have been numerous interesting studies that indicate that a significant percentage of users will trade their password for a candy bar, and the Anna Kournikova worm showed us that nearly 1/2 of humanity will click on anything purporting to contain nude pictures of semi-famous females. If "Educating Users" is the strategy you plan to embark upon, you should expect to have to "patch" your users every week. That's dumb.

The real question to ask is not "can we educate our users to be better at security?" it is "why do we need to educate our users at all?" In a sense, this is another special case of "Default Permit" - why are users getting executable attachments at all? Why are users expecting to get E-mails from banks where they don't have accounts? Most of the problems that are addressable through user education are self-correcting over time. As a younger generation of workers moves into the workforce, they will come pre-installed with a healthy skepticism about phishing and social engineering.

Dealing with things like attachments and phishing is another case of "Default Permit" - our favorite dumb idea. After all, if you're letting all of your users get attachments in their E-mail you're "Default Permit"ing anything that gets sent to them. A better idea might be to simply quarantine all attachments as they come into the enterprise, delete all the executables outright, and store the few file types you decide are acceptable on a staging server where users can log in with an SSL-enabled browser (requiring a password will quash a lot of worm propagation mechanisms right away) and pull them down. There are freeware tools like MIMEDefang that can be easily harnessed to strip attachments from incoming E-mails, write them to a per-user directory, and replace the attachment in the E-mail message with a URL to the stripped attachment. Why educate your users how to cope with a problem if you can just drive a stake through the problem's heart?

When I was CEO of a small computer security start-up we didn't have a Windows system administrator. All of the employees who wanted to run Windows had to know how to install it and manage it *themselves*, or they didn't get hired in the first place. My prediction is that in 10 years users that need education will be out of the high-tech workforce entirely, or will be self-training at home in order to stay competitive in the job market. My guess is that this will extend to knowing not to open weird attachments from strangers.

#6) Action is Better Than Inaction

IT executives seem to break down into two categories: the "early adopters" and the "pause and thinkers." Over the course of my career, I've noticed that *dramatically* fewer of the "early adopters" build successful, secure, mission-critical systems. This is because they somehow believe that "Action is Better Than Inaction" - i.e.: if there's a new whizzbang, it's better to install it *right now* than to wait, think about it, watch what happens to the other early adopters, and then deploy the technology once it's fully sorted-out and has had its first generation of experienced users. I know one senior IT executive - one of the "pause and thinkers" whose plan for doing a wireless roll-out for their corporate network was "wait 2 years and hire a guy who did a successful wireless deployment for a company larger than us." Not only will the technology be more sorted-out by then, it'll be much, much cheaper. What an utterly brilliant strategy!

There's an important corollary to the "Action is Better Than Inaction" dumb idea, and it's that:
"***It is often easier to not do something dumb than it is to do something smart.***"
Sun Tzu didn't *really* write that in "*The Art of War*" but if you tell IT executives that he did, they'll take you much more seriously when you counsel a judicious, thoughtful approach to fielding some new whizzbang. To many of my clients, I have been counselling, "hold off on outsourcing your security for a year or two and then get recommendations and opinions from the bloody, battered survivors - if there are any."

You can see the "Action is Better Than Inaction" dumb idea all over corporate networks and it tends to correlate with senior IT managers that make their product-purchasing decisions by reading Gartner research reports and product glossies from vendors. If you find yourself in the chain of command of such a manager, I sincerely hope you've enjoyed this article because you're probably far better acquainted with dumbness than I am.

One extremely useful piece of management kung-fu to remember, if you find yourself up against an "early adopter" is to rely on your peers. Several years ago I had a client who was preparing to spend a ton of money on a technology *without testing it operationally*. I suggested offhandedly to the senior IT manager in charge that he should send one of his team to a relevant conference (in this case, LISA) where it was likely that someone with hands-on experience with the technology would be in attendance. I proposed that the manager have his employee put a message on the "meet and greet" bulletin board that read:
"Do you have hands-on experience with ***xyz*** from ***pdq.com***? If so, I'm authorized to take you to dinner at Ruth's Chris if you promise to give me the low-down on the product off the record. Contact, etc..." The IT manager later told me that a $200 dinner expense saved them over $400,000 worth of hellish technological trauma.

It really is easier to not do something dumb than it is to do something smart. The trick is, when you avoid doing something dumb, to make sure your superiors know you navigated around a particularly nasty sand-bar and that you get appropriate credit for being smart. Isn't that the ultimate expression of professional kung-fu? To get ***credit*** for ***not*** doing ***anything***?!

The Minor Dumbs

These dumb ideas didn't quite merit status as "The Dumb*est*" ideas in computer security, but they're pretty dumb and deserve mention in passing:

"We're Not a Target" - *yes, you are*. Worms aren't smart enough to realize that your web site/home network isn't interesting.
"Everyone would be secure if they all just ran <security-flavor-of-the-month>" - *no, they wouldn't.* Operating systems have security problems because they are complex and system administration is not a solved problem in computing. Until someone manages to solve system administration, switching to the flavor-of-the-month is going to be *more* damaging because you're making it harder for your system administrators to gain a level of expertise that only comes with time.
"We don't need a firewall, we have good host security" - *no, you don't*. If your network fabric is untrustworthy every single application that goes across the network is potentially a target. 3 words: Domain Naming System.
"We don't need host security, we have a good firewall" - *no, you don't*. If your firewall lets traffic through to hosts behind it, then you need to worry about the host security of those systems.
"Let's go production with it now and we can secure it later" - *no, you won't*. A better question to ask yourself is "If we don't have time to do it correctly now, will we have time to do it over once it's broken?" Sometimes, building a system that is in constant need of repair means you will spend years investing in turd polish because you were unwilling to spend days getting the job done right in the first place.
"We can't stop the occasional problem" - *yes, you can*. Would *you* travel on commercial airliners if you thought that the aviation industry took this approach with your life? I didn't think so.

Goodbye and Good Luck

I've tried to keep this light-hearted, but my message is serious. Computer security is a field that has fallen far too deeply in love with the whizzbang-of-the-week and has forsaken common sense. Your job, as a security practitioner, is to question - if not outright challenge - the conventional wisdom and the status quo. After all, if the conventional wisdom was working, the rate of systems being compromised would be going ***down***, wouldn't it?

mjr.
Morrisdale, PA Sept 1, 2005
(A big "thank you" goes to Abe Singer and Tina Bird for contributing a couple dumb ideas, and to Paul Robertson and Fred Avolio for acting as the test choir)