# *Security by Design: Ridiculous Excuses We've Heard*

*Source:"Ridiculous Excuses We've Heard," of Writing Secure Code, Second Edition, Appendix B*

Now we're going to take an irreverent look at some of the excuses we've heard over the years from developers, testers, and designers from various companies trying to weasel out of making security design changes or code fixes! The excuses are:

- No one will do that!
- Why would anyone do that?
- We've never been attacked.
- We're secure we use cryptography.
- We're secure we use ACLs.
- We're secure we use a firewall.
- We've reviewed the code, and there are no security bugs.
- We know it's the default, but the administrator can turn it off.
- If we don't run as administrator, stuff breaks.
- But we'll slip the schedule.
- It's not exploitable.
- But that's the way we've always done it.
- If only we had better tools.

Let's get started.

No one will do that!

Oh yes they will! I once reviewed a product and asked the team whether it had performed buffer overrun tests on data the product received from a socket it opened. The team indicated that no, they had not performed such testing because no one would want to attack the server through the socket. Surely, no one would want to send malicious data at them in an attempt to attack their precious service! I reminded the team of the number of scripts available to attack various remote procedure calls (RPCs), Named Pipes, and socket interfaces on numerous platforms and that these could be downloaded by script kiddies to attack servers on the Internet. I even offered to help their testers build the test plans. But no, they were convinced that no one would attack their application through the socket the application opened.

To cut a long story short, I created a small Perl script that handcrafted a bogus packet and sent it to the socket the product opened, thereby crashing their server! Not surprisingly, the team fixed the bug and added buffer overrun testing to their test plans!

This group's people were not being glib; they were simply naive. Bad people attack computers, servers and desktops included, every day. If you don't think it will happen to you, you should think again!

Why would anyone do that?

This is a variation of the first excuse. And the answer is simple: because people are out there trying to get you, and they do it because they want to see you suffer! Seriously, some people enjoy seeing others discomfited, and some people enjoy vandalizing. We see it every day in the physical world. People scribble on the side of

buildings, and, sadly, some people like to pick fights with others so that they can harm them. The same holds true in the digital world. The problem in the digital world is that potentially many thousands of would-be attackers can attack you anonymously.

To sum up people attack computer systems because they can!

We've never been attacked.

When people say this, I add one word: yet! Or, as a colleague once told me, because no one cares about your product! As they say in the investment world, Past performance is no indication of future returns. This is also true in computer and software security. All it takes is for one attacker to find a vulnerability in your product and to make the vulnerability known to other attackers, and then other attackers will start probing your application for similar issues. Before you know it, you have a half-dozen exploits that need fixing.

I spent some time working closely with some product developers who said they had never been attacked so they didn't need to worry about security. Before they knew what hit them, they had seven security vulnerabilities in six months. They now have a small team of security people working on their designs and performing code reviews.

When I went to high school in New Zealand, there was a somewhat dangerous road leading up to my school. The school notified the local council that a pedestrian crossing should be built to allow pupils to cross the road safely. The council refused, citing that no one had been hurt, so there was no need to build the crossing. Eventually, a child was badly hurt, and the crossing was built. But it was too late a child was already injured.

This excuse reminds me of getting people to perform backups. Most people do so only after they have lost data. As long as a person has lost no data, the person thinks he or she is safe. However, when disaster strikes, it's too late: the damage is done.

The moral of this story is that bad things do happen and it's worthwhile taking preventive action as soon as possible. As my grandmother used to say to me, An ounce of prevention is worth a pound of cure.

We're secure we use cryptography.

Cryptography is easy from an application developer's perspective; all the hard work has been done. It's a well-understood science, and many operating systems have good cryptographic implementations. People make two major cryptographic mistakes, however:

- They design their own encryption algorithms.
- They store cryptographic keys insecurely.

If you've designed your own encryption algorithm, you're not using cryptography. Instead, you're using a poor substitute that probably will be broken.

If you insecurely store the keys used by the encryption system, you are also not using cryptography. Even the best encryption algorithm using the biggest keys possible is useless if the key is easily accessible by an attacker.

Don't create your own encryption algorithms. Use published protocols that have undergone years of public scrutiny.

We're secure we use ACLs.

Many resources in Windows NT, Windows 2000, and Windows XP can be protected using access control lists (ACLs). A good, well-thought-out ACL can protect a resource from attack. A bad ACL can lead to a sense of false security and eventually attack.

On a number of occasions I've reviewed applications that the developers claim use ACLs. On closer investigation, I found that the ACLs were Everyone (Full Control). In other words, anyone that's what Everyone means! can do anything that's what Full Control means! to this object. So the application does indeed include an ACL, but Everyone (Full Control) should not be counted because it's not secure.

We're secure we use a firewall.

This is another great excuse. I've heard this from a number of Microsoft clients. After looking at a client's Web-based architecture, I realize the client has little in the way of security measures. However, the client informs me that they've spent lots of money on their firewall infrastructure and therefore they are safe from attack. Yeah, right! Firewalls are a wonderful tool, but they are only part of the overall security equation.

Further examination of their architecture shows that just about everything is Web-based. This is worrisome. A firewall is in place, but many attacks come through the HTTP port, port 80, which is wide open through the firewall. It doesn't matter that there's a firewall in place a multitude of attacks can come through the firewall and straight into the Web server!

The client then mentions that they can inspect packets at the firewall, looking for malicious Web-based attacks. Performance issues aside, I then mention that I can use SSL/TLS to encrypt the HTTP traffic now the client cannot inspect the data at the firewall.

Firewalls are a wonderful tool when used correctly and as part of an overall security solution, but by themselves they don't solve everything.

We've reviewed the code, and there are no security bugs.

This is another of my favorite excuses. If you don't know what a security bug looks like, of course there are no security bugs! Can you certify a Boeing 747-400 for flight worthiness? Sure, we all can! It's got a bunch of wheels, two wings that droop a little (so they must be full of fuel), four engines, and a tail. It's good to go, right? Not by a long shot. There's a great deal more to check on any airplane to verify that it's safe, and it takes someone who knows what to look for to do the job correctly. The same holds true for reviewing code for security issues. You need to have the code reviewed by one or more people who understand how attackers attack code, what constitutes secure code, and what coding mistakes people make that lead to security vulnerabilities.

I remember performing a code review for an unreleased product. The specifications looked good, the small team consisted of high-caliber developers, and the test plans were complete. Now it was time to look at the code itself. Before the meeting started, the lead developer told me that the meeting was a waste of time because they had already performed code reviews looking for security issues and had found nothing. I suggested we have the meeting anyway and decide whether to continue after forty-five minutes. Suffice it to say, I found about 10 security bugs in twenty minutes, the

meeting continued for the three-hour duration, and a lot of people learned a great deal that day!

There is a corollary to this excuse: open source. Now, I have no intention of getting into a religious debate about open-source code. But software being open source does not mean it is more secure, most people simply just don't know what to look for. Actively looking at source code is a good thing, so long as you know what to look for and how to fix it. This is part of what David and I do at Microsoft: we review lots of code, and we know what to look for. We also act like bulldogs and make sure the bugs are fixed! It's a fun job!

We know it's the default, but the administrator can turn it off.

OK, let's cut to the chase administrators don't turn stuff off for five reasons:

- They often don't know what to turn off.
- They don't know how to turn it off.
- They don't know what will break if they do turn it off.
- They have perfectly stable systems why change things?
- They have no time.

Which leaves only one viable solution: design, build, test, and deploy systems that have practical yet secure defaults. Turning on a feature that could render a system vulnerable to attack should be a conscious decision made by the administrator.

We learned this hard lesson in Microsoft Internet Information Services (IIS) 5; IIS 6 now has most features turned off by default. If you want to use many features, you have to enable them. This is totally reasonable in systems that are susceptible to attack basically, any system that opens a socket!

If we don't run as administrator, stuff breaks.

I've had lots of conversations over the years that go like this. Me: What stuff breaks? Client: Security stuff breaks! Me: What do you mean security stuff breaks'? Client: If we don't run as admin, we get access denied' errors. Me: Do you think there's a good reason for that?

This is an example of not understanding the principle of least privilege. If you get an access denied, simply run the code as an administrator, or as local system, and the error goes away! This is rarely a good idea. Most day-to-day tasks do not require running as administrator. You should run with just the right privilege to get the job done and no more.

There is a side issue, however. Sometimes systems are written poorly, and people must run as administrator simply to get the job done. As software developers, we need to move away from this and support running with least privilege when nonprivileged tasks are performed. It's not that difficult to achieve, and it's a worthy goal!

I'm not a member of the Local Administrators group on my laptop and haven't been for three years. Granted, when I'm building a new machine, I do add myself to the Local Administrators group and then I remove myself. Everything works fine. When I want to run an administrative tool, I simply run with alternate credentials.

But we'll slip the schedule!

Thankfully, we are hearing this excuse less often these days as people realize the importance of delivering secure applications. However, in the bad old days, it was common to hear team leads drone on about it. Many development teams look at the effort required to make their product secure and realize that changing the way the product works means adding six months to the schedule. Look, you either pay now or pay later, and it's much more expensive in the future. Not only is it expensive for you, but it's expensive for all your customers. If you do ship something full of nasty security holes, you're going to stay in Security Patch Purgatory for a long time. So, add some time into your development process to make sure that appropriate steps are taken to design, build, test, and document a secure system. Treat security as a feature of the product, and stop whining!

It's not exploitable!

Both David and I have heard this a great deal, and it usually involves a code defect and the ensuing argument over whether the defect can be exploited by an attacker. The pattern is common: It would take 30 minutes to fix the defect or 10 days to analyze the bug and create an exploit to prove it's exploitable. No one is willing to spend ten days building an exploit, therefore it cannot be proven to be exploitable, therefore it's not exploitable, therefore the bug is not fixed. This is wrong. We agree that all bugs should be triaged accordingly. You might consider fixing a bug in the next release when the issue affects one person in a million, only when the Moon occults Saturn, and making the fix now would render the other 999,999 people's computers inoperable. However, security issues are different: if you spot a security flaw and the chance of regressions is slim, you should simply fix it. Don't wait for someone outside of your company to prove to you that the defect is indeed exploitable.

But that's the way we've always done it.

It doesn't matter how you did things in the past. Over the last few years, the Internet has become much more hostile and new threats emerge weekly. Frankly, this excuse probably indicates your products are hopelessly insecure and need some serious analysis and a great number of code changes to fix your old ways. Can you imagine your doctor prescribing a course of leeches to fix your headaches because that's the way we've always done it?

If only we had better tools

Yeah, sure! I've heard this excuse a couple of times; the problem is you cannot abdicate the responsibility of building secure software because of a tool. Tools can only do so much, and frankly, most tools are really dumb! When asked what tools he uses, one of the best code reviewers I know says, Notepad and my head.

Tools can help leverage the process, but sloppy developers using the best tools in the world still produce sloppy code. There's simply no replacement for good coding skills. The best developers know that tools are nothing more than a useful aid.