Complying with compliance

Blowing it off is not an option.

“Hey, compliance is boring. Really, really boring. And besides, I work neither in the financial industry nor in health care. Why should I care about SOX and HIPAA?”

Yep, you’re absolutely right. You write payroll applications, or operating systems, or user interfaces, or (heaven forbid) e-mail servers. Why should you worry about compliance issues?

Compliance is about that most boring and incomprehensible topic: laws and regulations. And it’s not all that well defined. Compliance encompasses everything from making sure that personal finance information doesn’t fall into the wrong hands to enforcing sexual harassment policies.

There are a few of us who can blow off this topic. Some of us really do write screen-saver software that doesn’t interact with anything else. I know one guy who builds robotic toys, which have safety concerns, but not *compliance* issues, in the sense of this article at least. Many researchers need their code to run only long enough to get their theses written, at which point the code can go to the great bit-bucket in the sky. Most of us, however, do have to worry about compliance.

Be you an application programmer, systems developer, or systems administrator, you’re probably going to run up against compliance sooner or later. Probably sooner.

MOST OF US LIVE IN THE REAL WORLD

It’s often said that writing software would be so much easier if only we didn’t have to deal with users. And so it is. The reality, however sad it may be, is that most of us must or want to write software that works in some-

ERIC ALLMAN, SENDMAIL

thing like the “real” world. Open source? Well, why open source your code in the first place if you don’t expect other people to use it?—and you can’t predict how they are going to use it. Mail programs, operating systems, network stacks, application libraries? They might (read: will) be used in the financial industry, healthcare, government, you name it (that is, if you succeed; if you plan on everything you do failing, you can ignore all of this). That means you should know at least the basics.

A good rule of thumb is probably this: If your code might be used in a business setting and will ever directly or indirectly interact with the public, or if it will be used by a company in a regulated industry or by a publicly traded company, you will probably need to come to grips with compliance. This is interpreted broadly; for example, authentication and authorization modules will probably be impacted even though they don’t explicitly manipulate business data, since they can be used in a sensitive context. There are the obvious cases, of course: If you are writing financial applications, you will need to comply with SEC regulations; if your application manages, processes, stores, or in any way touches business-sensitive data for a public company in the United States, you must pay attention to SOX (Sarbanes-Oxley Act of 2002); if your application will be run in Europe, you should be aware of appropriate EC (European Commission) regulations; there are international accords such as Basel II providing capital regulations for the banking industry; health patient record management systems need to comply with HIPAA (Health Insurance Portability and Accountability Act); and so forth down the line.

Suppose that you write a database library. Financial and health records are maintained in databases, so you probably need to think about how to achieve compliance across multiple regulations. If you work on anything related to communications—whether networking, e-mail, instant messaging, or Web applications—your code will probably be used to transfer sensitive data at some point.

COMPLIANCE COMPONENTS

What do you have to worry about if you are writing software that might someday, somehow be used in a regulated company?

Privacy. One of the most basic elements of compliance regulations is to keep private records private. It’s almost impossible to open a newspaper these days without reading about some organization that has had data lost or stolen. Besides creating compliance problems, these incidents are expensive and embarrassing for the organization.

Information to be kept private includes account numbers, social security numbers, financial data (account balances, etc.), data on treatments and prescriptions, financial vehicles, addresses and phone numbers, you name it. Perhaps a good metric would be to ask yourself, “Would I mind if this data were published about me? Would my friends mind if it were about them? Would it help someone who was trying to steal my identity?”

Just to make things extra complex, the privacy regulations vary from place to place, occasionally in incompatible ways. There’s more than just the U.S. federal government to consider. Some states have local laws, and, of course, the European Union has its own regulations. For example, *The Times* (June 10, 2006) reported that the Equal Opportunities Commission in the U.K. advised that all companies need to scan e-mail for possible sexual harassment.

Examples of issues to think about include:

- Am I storing data that I don’t really need to store?
- Am I encrypting all the data that I really do need to store?
- Have I taken adequate precautions to ensure that only people with proper authorization can access this data?
- Are my backups encrypted?

Accountability. Companies are held accountable for inappropriate use of data or systems. For example, there was a case where analysts at a large financial house were working in concert with the traders at the same company to “pump and dump” certain stocks. An e-mail paper trail showed that some analysts had been publicly touting a particular stock while privately referring to it as “a dog.” The point is not that you should never save your e-mails so that you can’t be caught, but rather to make sure the trouble doesn’t occur in the first place. Many companies have created “Chinese Walls” to prevent people on opposite sides of the house from exchanging sensitive information.

Auditability. Increasingly, regulators are demanding that information systems leave a virtual paper trail of everything they do in the event that information does leak. For example, many financial institutions are required to keep copies of everything that goes in and out of the company: physical mail, e-mail, instant messages, recordings of telephone calls, etc.

If information does leak, you want to know who leaked it and how broadly it was disseminated. You also have to work on the assumption that some employees are acting inappropriately, either through ignorance or malicious intent; it’s important to be able to find them and train them or fire them, as appropriate for the transgres-

sion. An audit trail can also be useful to protect yourself: If you find yourself in court having to assure a jury that you did *not* e-mail a hot stock tip to a customer, you want a complete record of all the e-mail that you *did* send.

It's worth pointing out that auditability does not mean "retain all data forever." Besides being impractical, this can have policy and legal implications. Instead of deleting data randomly, however, a company should have a clear retention policy that gets followed throughout the organization.

Policies. Companies should have clear, written policies for how data is handled. Of course, it's not your job to create or write those policies, but it is your job to make sure that your code can adapt to different policies from different companies.

Some of these policies may transcend the obvious regulations. Some companies have been successfully sued because their e-mail systems were used to convey material that was considered to be sexual harassment. This means that companies increasingly want to scan internal mail for words that might indicate a problem. For example, e-mails with the words *[deleted -ed.]*, *[deleted]*, or *[deleted]* probably shouldn't be transmitted through corporate e-mail—and it's the job of the e-mail system to catch these.

Manageability. Compliance generally requires that management be able to determine the status of information. Code should be written to adapt to monitoring—for example, by including extensible support for MIBs (management information bases). Applications and libraries should be designed with these requirements in mind.

Documentation. Being able to document what you do is often more important than what you document. In fact, many compliance regulations do not require that you do something specific, but that your company clearly documents what it does, actually follows those policies, and makes those policies available as appropriate.

For example, you've probably received privacy policies from the financial institution of your choice. Many of

these read, more or less, as follows: "We can give your data to any of our valuable marketing partners at our discretion." But the important part is that they have documented it. Similarly, your code should have clear, documented interfaces and behavior.

SUMMARY

A fair amount of what I've said here is utopian. Unfortunately, most backup tapes are not encrypted (but probably should be), and even with the best of intentions a lot of code is under-documented—and I certainly have to include much of my own code in that. In all likelihood, however, these types of compliance requirements are going to become more stringent over time, not less, and these will become larger issues.

The bad news is that these regulations are huge, complex, and frankly inscrutable to anyone who doesn't have a law degree in a specialized area. The good news is that the vast majority of these regulations are not about the computerized portion of the problem and, hence, aren't going to be your problem. Q

Disclaimer: I am not a compliance expert, just another technologist who has been dragged into this world kicking and screaming. This is just a bit of what I've learned in this very confusing area.—EA

LOVE IT, HATE IT? LET US KNOW

feedback@acmqueue.com or www.acmqueue.com/forums

ERIC ALLMAN is the cofounder and chief science officer of Sendmail, one of the first open source-based companies. He was previously the lead programmer on the Mammoth Project at the University of California at Berkeley. This was his second incarnation at Berkeley, as he was the chief programmer on the INGRES database management project. In addition to his assigned tasks, he got involved with the early Unix effort at Berkeley. His first experiences with Unix were with 4th Edition. Over the years, he wrote a number of utilities that appeared with various releases of BSD, including the -me macros, tset, trek, syslog, vacation, and, of course, sendmail. He spent the years between the two Berkeley incarnations at Britton Lee (later Sharebase) doing database user and application interfaces, and at the International Computer Science Institute, contributing to the Ring Array Processor project for neural-net-based speech recognition. He also coauthored the "C Advisor" column for *Unix Review* for several years. He was a member of the board of directors of the Usenix Association.

© 2006 ACM 1542-7730/06/0900 \$5.00