Logging on with KV

Some readers read Kode Vicious for the humor. Others read him for the biting critique of life in the software development trenches. But beneath his entertaining persona lies the unifying reason why loyal readers seek him out every month: his valuable advice on real problems that all programmers face. Although space limitations often prevent him from giving full treatment to any one issue, KV is sure at the very least to get you thinking in the right direction...and sometimes, that's all you need.

Dear KV,

I've been stuck with writing the logging system for a new payment processing system at work. As you might imagine, this requires logging a lot of data because we have to be able to reconcile the data in our logs with our customers and other users, such as credit card companies, at the end of each billing cycle, and we have to be prepared if there is any argument over the bill itself.

I've been given the job for two reasons: because I'm the newest person in the group and because no one thinks writing yet another logging system is very interesting. I've not gotten a lot of help from the other people on the team, who claim to have "written far more logging systems in their time than they want to think about." Do you have any advice on doing up a proper logging system?

Logged Out

Dear Logged Out,

If so many of your teammates have written logging systems before, then how come you're not using those? Perhaps your teammates are lying to you and have never written a single line of a logging system, or perhaps—and

Got a question for Kode Vicious? E-mail him at kv@acmqueue.com—if you dare! And if your letter appears in print, he may even send you a Queue coffee mug, if he's in the mood. And oh yeah, we edit letters for content, style, and for your own good!

A koder with attitude, KV ANSWERS

YOUR QUESTIONS.

MISS MANNERS HE AIN'T.

I suspect this is probably more likely—they tried and their systems sucked rocks. Or perhaps I'm just being cynical.

It turns out that writing a good logging system, like writing any good piece of software, is both difficult and rare. Many of your decisions are going to depend on the requirements put on the data you're logging, and since you're logging financial transactions, you have a lot of requirements, some of which must include the ability to keep the data private, audit the log for errors, and verify that the data contained in the log has not been tampered with.

Data privacy is now a big deal in our industry. It's too bad that it wasn't a big enough deal to the companies made famous in the last few years for breaching private data, such as ChoicePoint, Bank of America, Wells Fargo, and Ernst and Young, but they're all smarting for it now. Personal data breaches are now such a big problem that several governments have enacted strong legislation to punish those offenders, and I think you would like to avoid such punishment. I know I would.

The best way to keep data private is not to store it at all. Storing data makes it possible to breach it, which seems obvious, but then again every time I think something is obvious I wind up reading a news item that tells me, no, not obvious enough. Only keep whatever data you need to back up whatever claim you need to make, and don't keep data for too long. Most financial institutions have limits on how long they'll keep data. Follow the relevant ones for your product to the letter, and don't keep anything a second longer than you need it.

Once you've winnowed down the list of things you actually need to keep in the log, decide which ones can be blinded, which ones must be encrypted, and which can be left in the open. Blinding data means that it is destroyed, but in a way that makes it unique. A hash function is a great way to do this. Given any input, a good hash function produces unique, seemingly random, output. Consider the following example using the MD5 program on my Mac:

```
? md5 -s "1234 5678 9012 3456"
MD5 ("1234 5678 9012 3456")
= d135e2aaf43ba5f98c2378236b8d01d8
```

```
? md5 -s "1234 5678 9012 3457"
MD5 ("1234 5678 9012 3457")
= 0c617735776f122a95e88b49f170f5bf
```

Given two strings, which look like fake credit card numbers, where only one digit is different in one position, the MD5 program produces what looks like two different random numbers. If you can find a pattern in these, please contact your local MI6 or equivalent, as they have a job for you in the signals department.

Not only are these two numbers seemingly random, but they are also unique, which means they make a fine primary key for using in your data logging. Each log entry

I am amazed at the number of people who go to great lengths to **encrypt data** but then just chuck it all, unceremoniously, **in plain form**, into the logs.

with these numbers uniquely identifies the credit card, but someone reading the log cannot figure out the original credit card number from the hash. Blinding can be used on all kinds of data, but it's definitely good to use it on things that if they were stolen or compromised could be used by others.

If there is data that you absolutely must be able to use again in its original form—that is, it cannot be blinded—then it's time to start encrypting, at least if that data is valuable. I am amazed at the number of people who go to great lengths to encrypt data in their databases and live systems but then just chuck it all, unceremoniously, in plain form, into the logs. I guess I should stop being amazed, but it's preferable to banging my head on the desk, wall, floor, or the engineer in question.

What kind of data might need to be kept secret in your logs? An exhaustive list isn't possible, but certainly personal details such as the person's full name, address, phone number, mobile number, and e-mail address are a good start. While you're at it, the amounts paid, locations of payments, and other payment specifics should also be kept secret, as they make your logs a juicy target for people trying to dig up financial data on your company. You might be asking, "Well, what's left?" I would have to say that in a financial system, probably not a lot, but I'm sure there is data around for debugging purposes that might be OK to go into the log in plain form. For example, the time the entry was made is probably not going to be secret.

Now that you've eliminated all extraneous data, blinded what you could, and likely encrypted most of the rest, you have to make sure that the log itself is secure from tampering. You will need to do two things to prevent tampering: sign the entries and sign the log, each with a different key. The entries are signed to ensure their validity, and the whole log is signed to make sure that no one has added or removed entries by hand. The reason for using two different keys is that two different people should have those keys, thereby requiring collusion to violate the security of the system. It's also a good idea to change your keys regularly, so that if a key is stolen, the amount of data that is exposed is minimized.

There are many other things to touch upon with a logging system, such as where the data is stored, how it may or may not be moved across a network, when the logs need rotating, and how to write tools to analyze and read the log. But what I've presented here are the basics of making a logging system that, I hope, doesn't suck and doesn't make it trivial to violate the privacy of your users and land your company on the front page of the news. Oh, and one last piece of advice: Don't leave the logs on a laptop in your car. Obvious? Sure, it's obvious.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who has made San Francisco his home since 1990.

© 2006 ACM 1542-7730/06/0600 \$5.00