Commentary on Password Hashing:

This is yet another explanation of a programming concept. I seem to be rather fond of these. I'm not sure whether or not it's well-explained, but you can read about it here if you'd like.

When I felt pressured to fill up pages for the writer's notebook, this is what I did. It felt like a cheaty way out; there's no way to argue that it doesn't have a lot of dense content, but it was still pretty easy (and enjoyable) to write. Mr. Petrushun never complained about my constant attempts to teach him things that he didn't care about, so I managed to make it through the entire year doing this.

I'm content with the situation. The only problem is that it ended with the necessity to apologize to everyone who reads my portfolio in advance.

It's worth it.

Writer's Notebook: Password Hashing 6 January 2013

As a website developer, you're required to act as though the contents of your private database are publicly available. An attacker might compromise the contents of your database at any time, so you need to store the data very carefully.

That means it's a *really* bad idea to store user information in the database like this:

user	password	email
george	ilovetacos	george@gmail.com
jim	ihategeorge	jim@yahoo.com

If an attacker gains access to the database, he can log into anyone's account. If users are stupid enough to reuse passwords, he can also log into their email accounts.

How do you fix that, then? It's kind of common sense that you need to store passwords if you want to let users log in, right?

As it turns out, it's really easy to keep passwords out of the database while still effectively authenticating passwords.

There exist functions called hashing functions that will, provided an input, produce a hash. The hash is a string of numbers or text (again, it's a bunch of ones and zeroes that you can represent however you'd like) that *cannot* be reversed; knowing a datum's hash will give you no insight as to what datum produced that hash, in a perfect hash function.

So, really, that's all that websites have to do. When you sign up, your password is run through a hash function and the hash is stored. Then, when you log in later, the password you type is hashed and the result hash is compared to what is stored in the database.

Hashing is useful and powerful, but only when it's done correctly. A popular hash algorithm used today is md5. md5 has a few big issues, though.

md5 is *fast*. md5 is *optimised* to be fast. md5 wasn't actually designed for password hashing. It was designed to create checksums for files to verify their integrity. It's very useful for that (though it's no longer recommended to hash RSA certificates with md5, because successful collision attacks have been performed—that, however, is beyond the scope of this writing), but there's a problem with having speed in your password hashing algorithm.

If a hacker can access the hashed password, he can brute-force every single password