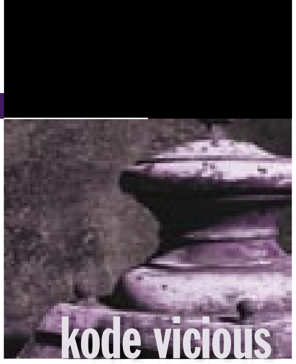

Kode Vicious Gets Dirty

A koder with attitude, KV ANSWERS YOUR QUESTIONS. MISS MANNERS HE AIN'T.

Articles in *Queue* focus on the difficult and challenging aspects of using new technologies. Quite often these challenges involve large-scale system and architectural issues that arise when deploying and integrating new software. But what about all that low-level, nitty-gritty koding stuff? Well, that's where Kode Vicious comes in. KV actually enjoys helping people with these dirty details. And not just the technical dirt—no one kodes in a vacuum, and consequently, software development is rife with interpersonal conflict and organizational breakdown. There, too, our resident kode maven offers up his hard-won expertise and strong opinions for the benefit of koders everywhere. His advice may not always be pretty, but it's sure to solve your problems. Unfortunately, you're on your own with any resulting medical or legal bills.

Dear Kode Vicious,
I am a new Webmaster of a (rather new) Web site in my company's intranet. Recently I noticed that although I have implemented some user authentication (a start *.asp page linked to an SQL server, having usernames and passwords), some of the users found out that it is also possible to enter a rather longer URL to a specific page within that Web site (instead of entering the homepage), and they go directly to that page without being authenticated (and without their login being recorded in the SQL database). It makes me wonder what solution you could advise me to implement to ensure that any and all Web accesses are checked and recorded by the Web server.

New Web Master

Dear NWM,
You have my deepest sympathies. Users, as I have noted before, are the bane of our existence. Those sneaky bastards will go around your systems every time to get the data they want, without paying your login system any heed. Now, there are many ways to deal with recalcitrant users, but unfortunately I can only discuss the ones that are legal here; the others, trust me, are far more enjoyable.

Not to be too cruel, but it seems from your description that you have created a superfluous authentication system that doesn't provide much for the users or for you. Users can and do go around your login page, and therefore you've just created more code without any real value, and valueless code is a real shame. You need to change your way of thinking about this problem before you can solve it.

Right now use of your authentication system is voluntary, and therefore easily bypassed, because you are not enforcing the authentication on each page that your users can see. In your letter you don't say that your system has any of the necessary features of an authentication system, such as:

1. What the authentication gives the users the right to do. For example can they read, modify, or create pages?
2. How users prove to the system that they were authenticated.
3. How users and the system agree that the users are who they say they are.

You have a system of Web pages, which you believe contain valuable information since you claim you want to protect them. Yet those pages have no protection from anyone who can work out or guess what the name of the link is?! That's just plain wrong. If you have information that must be protected, then you should definitely be protecting it.

One way to do so is to implement the features shown in the list—namely, that the users must prove who they are to your login system and then must prove they were authenticated and have a right to see the information whenever they want to read a page.

How do the users prove they have these rights? They must first talk to the login system to prove who they are.

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!*

Infinite time-outs on authentication cookies make those cookies very valuable to steal.

In your case you implemented a Web page that fronts a database of usernames and passwords.

As a quick aside, I hope you are storing only the hash of the password and not the raw text. A hash uniquely destroys the password so that if the password was “foo,” the resulting hashed value might be the number 5. Given the number 5, someone cannot get back to “foo,” but given “foo” and the same hash function, you will always get 5. When verifying a password, you’re really comparing the hashed values; the original string “foo” is never stored. Keeping a database of raw username and password pairs is a serious security hole, the kind of bug that makes KV want to make a big pot of programmer hash.

So, now the users can prove who they are by logging in with their usernames and passwords, but how can you satisfy feature 3 above? Each user submits a username and password to your system, but, well, so what? The real problem with your pages is that the pages themselves are not protected. If you want to force users to authenticate themselves, then each request to your Web server must include some piece of information that proves the user has been authenticated. If the server does not check requests to see if they come from authenticated users, then there is no point in having an authentication system at all. What should the user have to present to the system?

In the Web world the most common piece of information exchanged between a user—or more precisely the user’s Web browser—and a server is a cookie. A cookie is just a chunk of data that can be set by a server into a user’s Web browser. When the user is browsing within a particular domain, the server can look at the cookie and get information from it. In an authentication system, the server should set a cookie into the user’s browser, which is then checked on every subsequent access to the system in order to validate that this particular user has the right to use the system.

What should go into the cookie? Well, that’s mostly up to the data that you, as the system maintainer, wish to track, but two things are absolutely necessary to prevent the system from being abused. The first is that the cookie must have a digital signature. This can prevent the cookie from being manipulated by users to gain access when they should not. If someone were to find out what the format of your cookies is, and your cookies were not signed, then that person could just craft their own cookies and present them to the server, bypassing your authentication system.

The second necessary part of the cookie is a time-out, past which the cookie is no longer good and must be replaced. Infinite time-outs on authentication cookies makes those cookies very valuable to steal, because once acquired they can never be revoked. Picking the time-out, though, is a balancing act. Your users will want their authentication tokens to last as long as possible, perhaps for months, whereas to maintain control of your systems, you would prefer something much shorter, such as one hour. Finding the compromise between the permissive and the fascist time-out is beyond the scope of this article, and depends on your users and how much management backup you have to force them to behave in the way that you would like. I find that reminding management of how much money they’ll lose if users are able to leak and leach information from the system is very effective in getting shorter time-outs implemented. Management hates losing money like KV hates losing the keys to his liquor cabinet.

So, now you have a model of an authentication system, instead of just a system that happens to record users logging in when they feel like it. A users logs in, gets a cookie that is digitally signed to prevent tampering and that has a limited lifetime, and must then use that cookie to read any of the other pages. There are many ways to implement this, but that’s the general outline. There are plenty of examples of how to do this on the Internet, so, get back in there and fix this!

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.