10 Ways Developers Put Databases At Risk

Some of the most important database protection methods start with developers who tap into sensitive data stores

Sep 12, 2012 | 10:46 PM

By Ericka Chickowski, Contributing Writer, Dark Reading

Many of today's Web applications rely on enterprises' most sensitive data stores to keep order systems running, partner companies collaborating, and internal users in touch with important business information no matter where they are.

[Learn more about best practices for protecting your databases. See Strategies For Protecting Web-Facing Databases .]

While such easy access to business-critical data has greatly improved worker productivity and loosened the pocketbooks of customers, it has also opened up that data to considerable risk. Unfortunately, much of the risk is introduced by developers who lack the resources to code these applications without vulnerabilities that open databases to compromise -- be it time, money, education, or support from executives.

When these factors aren't in place, developers frequently make these mistakes:

1. **Trusting Input Way Too Much**

One of the top ways developers put databases at risk today is by leaving their applications open to SQL injection. And those SQLi vulnerabilities usually crop up when the developer puts too much trust in user input.

"Without validating that, say, a phone number field only accepts digits, a developer may be allowing hackers something analogous to terminal access to the database," says Konstantinos Karagiannis, principal consultant for BT Global Services' Ethical Hacking division. "For example, a single quote inserted in a form field may prematurely close a valid SQL query the application was supposed to make, allowing the attacker to build and submit a new query."

One of the top ways of addressing this problem is by parameterizing queries.

"Parameterized queries prevent attackers from changing the logic or code at the heart of the query and negate most SQL injection attacks," says Jacob West, CTO of Fortify Products at HP Enterprise Security. "Parameterized queries have been around almost as long as databases and, unlike many areas of security, using them is a best practice for performance and code maintainability as well."

But developers shouldn't stop there. Input should be sanitized and validated because sometimes parameterization is not a possibility. Even when it is, unvalidated input can be used for other nefarious purposes, such as cross-site scripting attacks.

"The oldest and worst sin is not sanitizing inputs, which is stripping out any characters not needed from the user input, like removing apostrophes or semicolons, before running a query against the database," says David Habusha, product vice president for GreenSQL.

2. Displaying Database Error Messages To End Users

When developers allow very specific error messages to pop up when something goes wrong with an application's SQL query, it may help with diagnostics, but it is also offering hackers a nice little window into the inner workings of the back-end database.

"This makes attackers' jobs easier by revealing important information about how the database is structured and how the application queries it," says Bill Karwin, principal consultant for Percona.

Error messages can give away clues as to the type of database an application is tapping into, the query structure and the underlying schema, Habusha says.

"This exploit opens the door for a blind SQL injection attack," he says. "Developers should display a generic error page instead."

3. Playing Fast And Loose With Passwords

For the sake of expediency, many developers play fast and loose with user passwords in a number of insecure ways. For example, they may hard-code passwords into the application.

"One of the most egregious database risks are hard-coded passwords and passwords in configuration files," says Tim Erlin, director of IT security risk and strategy for nCircle. "Both of these design choices rely on security through obscurity; they assume the attacker won't have access to this information, and therefore development doesn't have to concern themselves with the security of the files or the information in them."

Similarly dangerous is the practice of storing passwords in plain text.

"A very simple but all too common mistake is storing passwords in plain text, which involves not hashing the user inputted password, and not salting hashes, which entails adding random bits to the hash," says Daniel Jacobowitz, security research engineer for BeyondTrust.

Whichever flavor of poor password management, failing to build strong authentication into applications with a direct line into the database is akin to leaving the door to the bank vault wide open.

4. Making Every Connection SUPER

In the same vein, many developers put databases at risk by routing all connections from the application to the database through a "root" or some other SUPER user account. This is often a frequent problem associated with those applications with hard-coded passwords, as it is too difficult to institute proper privilege management in these cases.

"Ordinary application operations rarely need access granted by the 'super' privilege, and allowing applications to use the super-privileged user creates opportunities for unrestricted actions that are not appropriate," Karwin says. "All applications should connect to the database using lesser-privileged credentials."

5. Believing Stored Procedures Are THE Answer To SQLi

Many developers today believe that stored procedures are a reliable way to prevent SQL injection. Not so, says Dmitry Vyrostkov of DataArt.

"In fact, stored procedures do not prevent SQL injections if they contain vulnerabilities within their own code, or if they are invoked in an unsafe manner," he says.

Romain Guacher agrees that this is a myth that needs busting within the developer community.

"It's not true because string concatenation can happen inside stored procedures," says Gaucher, senior security researcher for Coverity. "Even stored procedure calls that are prepared statements can be vulnerable if dynamic queries are constructed and executed from inside the stored procedure itself."

6. Leaving Debug Code In Production

Just like you have to clean the kitchen up after cooking a really good meal, developers have got to clean up their code before putting things in production lest they leave back doors open into the database. One of the most commonly forgotten elements to cause such sloppy back doors is debug code left in production, Gaucher says.

"It's not unknown that developers put in a back door to enable database queries to be directly input for debugging purposes during development," he says. "Forgetting to remove such debug code in production is a silly but all too common mistake."

7. Implementing Shoddy Encryption

According to Anthony Moulton, senior software development engineer for Vigilant, the only thing worse than not using encryption is using it wrong because it gives the organization a false sense of security.

"The devil is in the details, whether it is a hashing function or an encryption routine," Moulton says. "If you don't know how to implement cryptology properly, delegate the task to an expert and do not take your application into production prematurely."

Developers should be especially careful of falling prey to hubris about their L33t skills in cryptology. Today's hackers love home-grown encryption schemes because they're usually done poorly, says Dan Brown, director of security research for Bit9.

"Home-grown encryption provides almost no value to experienced crackers and gives a false sense of security to organizations that deploy it," Brown says. "Ditto for organizations that misunderstand what threat scenarios their super-strong military-grade encryption is intended to address."

8. Putting Blind Trust In Third-Party Code

Using third-party code may save developers a lot of time, but where they shouldn't cut corners is in testing to ensure that a cut-and-paste isn't introducing vulnerable code to the application.

"Developers need to design with the understanding they will be responsible for threat analysis of the entire application, not just the code they've written themselves," Erlin says.

Similarly, when coders use development frameworks, they need to stay up on updates if they want to limit the number of vulnerabilities opening their databases to risk.

"People will use WordPress, Joomla, or any other type of application framework and not keep them properly updated with all the latest security patches," says Kenneth Pickering, development manager of security intelligence for Core Security. "Much like machines are vulnerable, so too are application frameworks."

9. **Naïvely Implementing RESTful Architecture**

Representational state transfer (REST) architectures might be a useful paradigm, but too many tools generate REST interfaces directly from the database, Karwin says.

"This couples the interface to the schema, and exposes information that attackers can use," he says. "Developers should design REST interfaces for a set of abstract application-specific resource types and resource-appropriate operations, not directly to the physical database tables and generic CRUD operations."

10. **Leaving Backup Database Copies Lying Around**

Many developers have been chided enough about not using live data in test environments that they've looked for alternative means of testing their applications. Unfortunately, a large subset of these coders will turn to backup databases instead.

"A site may protect the live database well, but do they protect their database backups similarly? Week-old data in a backup may be as damaging in the wrong hands as the live data," Karwin says. "Developers may use backups for testing their work against samples of production data, so development environments must follow security standards as carefully as production environments. Security policies should account for all copies of data, not just the online copy."