



Top 25 Most Dangerous Software Errors

SANS Institute 2011



Out of more than 700 ...

the most widespread and critical errors that can lead to serious vulnerabilities in software. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all.



25th place

Use of a One-Way Hash without a Salt

- Salt might not be good for your diet, but it can be good for your password security. Instead of storing passwords in plain text, a common practice is to apply a one-way hash, which effectively randomizes the output and can make it more difficult if (or when?) attackers gain access to your password database.



24th place

Integer Overflow or Wraparound

- In the real world, $255+1=256$. But to a computer program, sometimes $255+1=0$, or $0-1=65535$



23rd place

Uncontrolled Format String

- The mantra is that successful relationships depend on communicating clearly, and this applies to software, too. Format strings are often used to send or receive well-formed data. By controlling a format string, the attacker can control the input or output in unexpected ways



22nd place

URL Redirection to Untrusted Site

- Many web applications have implemented redirect features that allow attackers to specify an arbitrary URL to link to, and the web client does this automatically
- First, the victim could be automatically redirected to a malicious site that tries to attack the victim through the web browser. Alternately, a phishing attack could be conducted, which tricks victims into visiting malicious sites that are posing as legitimate sites. Either way, an uncontrolled redirect will send your users someplace that they don't want to go.



21st place

Improper Restriction of Excessive Authentication Attempts

- An often-used phrase is "If at first you don't succeed, try, try again." Attackers may try to break into your account by writing programs that repeatedly guess different passwords.



20th place

Incorrect Calculation of Buffer Size

- In languages such as *C*, where memory management is the programmer's responsibility, there are many opportunities for error. If the programmer does not properly calculate the size of a buffer, then the buffer may be too small to contain the data that the programmer plans to write - even if the input was properly validated.



19th place

Use of a Broken or Risky Cryptographic Algorithm

- You may be tempted to develop your own encryption scheme in the hopes of making it difficult for attackers to crack. This kind of grow-your-own cryptography is a welcome sight to attackers.



18th place

Use of Potentially Dangerous Function

- The programmer's toolbox is chock full of power tools, including library or API functions that make assumptions about how they will be used, with no guarantees of safety if they are abused.



17th place

Incorrect Permission Assignment for Critical Resource

- It's rude to take something without asking permission first, but impolite users (i.e., attackers) are willing to spend a little time to see what they can get away with. If you have critical programs, data stores, or configuration files with permissions that make your resources readable or writable by the world - well, that's just what they'll become.



16th place

Inclusion of Functionality from Untrusted Control Sphere

- The idea seems simple enough (not to mention cool enough): you can make a lot of smaller parts of a document (or program), then combine them all together into one big document (or program) by "including" or "requiring" those smaller pieces. This is a common enough way to build programs. Combine this with the common tendency to allow attackers to influence the location of some of these pieces - perhaps even from the attacker's own server - then suddenly you're importing somebody else's code.



15th place

Incorrect Authorization

- While the lack of authorization is more dangerous (see elsewhere in the Top 25), incorrect authorization can be just as problematic. Developers may attempt to control access to certain resources, but implement it in a way that can be bypassed. For example, once a person has logged in to a web application, the developer may store the permissions in a cookie. By modifying the cookie, the attacker can access other resources.



14th place

Download of Code Without Integrity Check


- if you download code and execute it, you're trusting that the source of that code isn't malicious. Maybe you only access a download site that you trust, but attackers can perform all sorts of tricks to modify that code before it reaches you. They can hack the download site, impersonate it with DNS spoofing or cache poisoning, convince the system to redirect to a different site, or even modify the code in transit as it crosses the network.



13th place

Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

- When you use an outsider's input while constructing a filename, the resulting path could point outside of the intended directory. An attacker could combine multiple ".." or similar sequences to cause the operating system to navigate out of the restricted directory, and into the rest of the system.



12th place

Cross-Site Request Forgery (CSRF)

- In cross-site request forgery, the attacker tricks a user into activating a request that goes to your site. Thanks to scripting and the way the web works in general, the user might not even be aware that the request is being sent. But once the request gets to your server, it looks as if it came from the user, not the attacker, who has essentially masqueraded as a legitimate user and gained all the potential access that the user has. This is especially handy when the user has administrator privileges, resulting in a complete compromise of your application's functionality.



11th place

Execution with Unnecessary Privileges

- Your software may need special privileges to perform certain operations, but wielding those privileges longer than necessary can be extremely risky. When running with extra privileges, your application has access to resources that the application's user can't directly reach.



10th place

Reliance on Untrusted Inputs in a Security Decision

- Software developers often rely on untrusted inputs and when these inputs are used to decide whether to grant access to restricted resources, trouble is just around the corner.
 - For any security checks that are performed on the client side, ensure that these checks are duplicated on the server side,



9th place

Unrestricted Upload of File with Dangerous Type

- You may think you're allowing uploads of innocent images (rather, images that won't damage your system - the Interweb's not so innocent in some places). But the name of the uploaded file could contain a dangerous extension such as .php instead of .gif, or other information (such as content type) may cause your server to treat the image like a big honkin' program.



8th place

Missing Encryption of Sensitive Data

- If your software stores sensitive information on a local file or database, there are ways for attackers to get at the file. They may benefit from lax permissions, exploitation of another vulnerability, or physical theft of the disk. You know those massive credit card thefts you keep hearing about? Many of them are due to unencrypted storage.



7th place

Use of Hard-coded Credentials

- Hard-coding a secret password or cryptographic key into your program is bad manners, even though it makes it extremely convenient - for skilled reverse engineers. While it might shrink your testing and support budgets, it can reduce the security of your customers to dust.



6th place

Missing Authorization

- Software faces similar authorization problems that could lead to more dire consequences. If you don't ensure that your software's users are only doing what they're allowed to, then attackers will try to exploit your improper authorization and exercise unauthorized functionality that you only intended for restricted users.



5th place

Missing Authentication for Critical Function

- Software may expose certain critical functionality with the assumption that nobody would think of trying to do anything but break in through the front door. But attackers know how to case a joint and figure out alternate ways of getting into a system.



4th place

Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')

- Cross-site scripting (XSS) is one of the most prevalent, obstinate, and dangerous vulnerabilities in web applications. It's pretty much inevitable when you combine the stateless nature of HTTP, the mixture of data and script in HTML, lots of data passing between web sites, diverse encoding schemes, and feature-rich web browsers. If you're not careful, attackers can inject Javascript or other browser-executable content into a web page that your application generates. Your web page is then accessed by other users, whose browsers execute that malicious script as if it came from you (because, after all, it *did* come from you). Suddenly, your web site is serving code that you didn't write.



Medal Round

Now the top three among Most Dangerous Software Errors

**Stay tuned: they could move up or down when
we rank them again ...**



3rd place

Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')

- Buffer overflows are Mother Nature's little reminder of that law of physics that says: if you try to put more stuff into a container than it can hold, you're going to make a mess. The scourge of C applications for decades, buffer overflows have been resistant to elimination. However, copying an untrusted input without checking the size of that input is the simplest error to make in a time when there are much more interesting mistakes to avoid.



2nd place

Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')

- Your software is often the bridge between an outsider on the network and the internals of your operating system. When you invoke another program on the operating system, but you allow untrusted inputs to be fed into the command string that you generate for executing that program, then you are inviting attackers to cross that bridge by executing their own commands instead of yours.



1st place

Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')

- These days, it seems as if software is all about the data: getting it into the database, pulling it from the database, massaging it into information, and sending it elsewhere for fun and profit. If attackers can influence the SQL that you use to communicate with your database, then suddenly all your fun and profit belongs to them. If you use SQL queries in security controls such as authentication, attackers could alter the logic of those queries to bypass security. They could modify the queries to steal, corrupt, or otherwise change your underlying data.