acunetix

2011 CWE/SANS Top 25 Most Dangerous Software Errors

Compliance Report

21 May 2025

Generated by Acunetix

Description

The 2011 CWE/SANS Top 25 Most Dangerous Software Errors is a list of 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.

The Top 25 list is a tool for education and awareness to help programmers to prevent the kinds of vulnerabilities that plague the software industry, by identifying and avoiding all-too-common mistakes that occur before software is even shipped. Software customers can use the same list to help them to ask for more secure software. Researchers in software security can use the Top 25 to focus on a narrow but important subset of all known security weaknesses. Finally, software managers and CIOs can use the Top 25 list as a measuring stick of progress in their efforts to secure their software.

The list is the result of collaboration between the SANS Institute, MITRE, and many top software security experts in the US and Europe. It leverages experiences in the development of the SANS Top 20 attack vectors (http://www.sans.org/top20/) and MITRE's Common Weakness Enumeration (CWE) (http://cwe.mitre.org/). MITRE maintains the CWE web site, with the support of the US Department of Homeland Security's National Cyber Security Division, presenting detailed descriptions of the top 25 programming errors along with authoritative guidance for mitigating and avoiding them. The CWE site contains data on more than 800 programming errors, design errors, and architecture errors that can lead to exploitable vulnerabilities.

Disclaimer

This document or any of its content cannot account for, or be included in any form of legal advice. The outcome of a vulnerability scan (or security evaluation) should be utilized to ensure that diligent measures are taken to lower the risk of potential exploits carried out to compromise data.

Legal advice must be supplied according to its legal context. All laws and the environments in which they are applied, are constantly changed and revised. Therefore no information provided in this document may ever be used as an alternative to a qualified legal body or representative.

This document was generated using information provided in "2010 CWE/SANS Top 25 Most Dangerous Software Errors", that can be found at http://cwe.mitre.org/top25/.

Scan

URL	http://sitalitbang.semarangkab.go.id
Scan date	21/05/2025, 14:53:39
Duration	14 seconds
Profile	Full Scan

Compliance at a Glance

This section of the report is a summary and lists the number of alerts found according to individual compliance categories.

- Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')(1)
No alerts in this category

- Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')(2)
No alerts in this category

- Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')(3)
No alerts in this category

- Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')(4)
No alerts in this category

- Missing Authentication for Critical Function(5)
No alerts in this category

- Improper Access Control (Authorization)(6)
No alerts in this category

- Use of Hard-coded Credentials(7)
No alerts in this category

- Missing Encryption of Sensitive Data(8)
No alerts in this category

- Unrestricted Upload of File with Dangerous Type(9)
No alerts in this category

- Reliance on Untrusted Inputs in a Security Decision(10)
No alerts in this category

- Execution with Unnecessary Privileges(11)
No alerts in this category

- Cross-Site Request Forgery (CSRF)(12)
No alerts in this category

- Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')(13)
No alerts in this category

- Download of Code Without Integrity Check(14)
No alerts in this category

- Incorrect Authorization(15)
No alerts in this category

- Inclusion of Functionality from Untrusted Control Sphere(16)
No alerts in this category

- Incorrect Permission Assignment for Critical Resource(17)
No alerts in this category

- Use of Potentially Dangerous Function(18)
No alerts in this category

- Use of a Broken or Risky Cryptographic Algorithm(19)
No alerts in this category

- Incorrect Calculation of Buffer Size(20)
No alerts in this category

- Improper Restriction of Excessive Authentication Attempts(21)
No alerts in this category

- URL Redirection to Untrusted Site ('Open Redirect')(22)
No alerts in this category

- Uncontrolled Format String(23)
No alerts in this category

- Integer Overflow or Wraparound(24)
No alerts in this category

- Use of a One-Way Hash without a Salt(25)
No alerts in this category

Compliance According to Categories: A Detailed Report

This section is a detailed report that explains each vulnerability found according to individual compliance categories.

(1)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. They'll even steal data one byte at a time if they have to, and they have the patience and know-how to do so.

No alerts in this category.

(2)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 into a land of riches by executing their own commands instead of yours.

No alerts in this category.

(3)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 remarkably 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. That's why this type of buffer overflow is often referred to as "classic." It's decades old, and it's typically one of the first things you learn about in Secure Programming 101.

No alerts in this category.

(4)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. The attacker can use a variety of techniques to get the input directly into your server, or use an unwitting victim as the middle man in a technical version of the "why do you keep hitting yourself?" game.

No alerts in this category.

(5)Missing Authentication for Critical Function

In countless action movies, the villain breaks into a high-security building by crawling through heating ducts or pipes, scaling elevator shafts, or hiding under a moving cart. This works because the pathway into the building doesn't have all those nosy security guards asking for identification. 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.

No alerts in this category.

(6)Improper Access Control (Authorization)

Suppose you're hosting a house party for a few close friends and their guests. You invite everyone into your living room, but while you're catching up with one of your friends, one of the guests raids your fridge, peeks into your medicine cabinet, and ponders what you've hidden in the nightstand next to your bed. 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.

No alerts in this category.

(7)Use of Hard-coded Credentials

Hard-coding a secret password or cryptograpic 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. If the password is the same across all your software, then every customer becomes vulnerable if (rather, when) your password becomes known. Because it's hard-coded, it's usually a huge pain for sysadmins to fix. And you know how much they love inconvenience at 2 AM when their network's being hacked - about as much as you'll love responding to hordes of angry customers and reams of bad press if your little secret should get out. Most of the CWE Top 25 can be explained away as an honest mistake; for this issue, though, customers won't see it that way. Another way that hard-coded credentials arise is through unencrypted or obfuscated storage in a configuration file, registry key, or other location that is only intended to be accessible to an administrator. While this is much more polite than burying it in a binary program where it can't be modified, it becomes a Bad Idea to expose this file to outsiders through lax permissions or other means.

No alerts in this category.

(8)Missing Encryption of Sensitive Data

Whenever sensitive data is being stored or transmitted anywhere outside of your control, attackers may be looking for ways to get to it. Thieves could be anywhere - sniffing your packets, reading your databases, and sifting through your file systems. If your software sends sensitive information across a network, such as private data or authentication credentials, that information crosses many different nodes in transit to its final destination. Attackers can sniff this data right off the wire, and it doesn't require a lot of effort. All they need to do is control one node along the path to the final destination, control any node within the same networks of those transit nodes, or plug into an available interface. If your software stores sensitive information on a local file or database, there may be other 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.

No alerts in this category.

(9)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. So, instead of seeing the latest paparazzi shot of your favorite Hollywood celebrity in a compromising position, you'll be the one whose server gets compromised.

No alerts in this category.

(10)Reliance on Untrusted Inputs in a Security Decision

In countries where there is a minimum age for purchasing alcohol, the bartender is typically expected to verify the purchaser's age by checking a driver's license or other legally acceptable proof of age. But if somebody looks old enough to drink, then the bartender may skip checking the license altogether. This is a good thing for underage customers who happen to look older. Driver's licenses may require close scrutiny to identify fake licenses, or to determine if a person is using someone else's license. Software developers often rely on untrusted inputs in the same way, and when these inputs are used to decide whether to grant access to restricted resources, trouble is just around the corner.

No alerts in this category.

(11)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. For example, you might intentionally launch a separate program, and that program allows its user to specify a file to open; this feature is frequently present in help utilities or editors. The user can access unauthorized files through the launched program, thanks to those extra privileges. Command execution can happen in a similar fashion. Even if you don't launch other programs, additional vulnerabilities in your software could have more serious consequences than if it were running at a lower privilege level.

No alerts in this category.

(12)Cross-Site Request Forgery (CSRF)

You know better than to accept a package from a stranger at the airport. It could contain dangerous contents. Plus, if anything goes wrong, then it's going to look as if you did it, because you're the one with the package when you board the plane. Cross-site request forgery is like that strange package, except 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. This might not seem like a big deal, but the attacker 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. When combined with XSS, the result can be extensive and devastating. If you've heard about XSS worms that stampede through very large web sites in a matter of minutes, there's usually CSRF feeding them.

No alerts in this category.

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

While data is often exchanged using files, sometimes you don't intend to expose every file on your system while doing so. 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.

No alerts in this category.

(14)Download of Code Without Integrity Check

You don't need to be a guru to realize that 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. This scenario even applies to cases in which your own product downloads and installs its own updates. When this happens, your software will wind up running code that it doesn't expect, which is bad for you but great for attackers.

No alerts in this category.

(15)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. Alternately, the developer might perform authorization by delivering code that gets executed in the web client, but an attacker could use a customized client that removes the check entirely.

No alerts in this category.

(16)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. In these Web 2.0 days, maybe it's just "the way the Web works," but not if security is a consideration.

No alerts in this category.

(17)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. While this issue might not be considered during implementation or design, sometimes that's where the solution needs to be applied. Leaving it up to a harried sysadmin to notice and make the appropriate changes is far from optimal, and sometimes impossible.

No alerts in this category.

(18)Use of Potentially Dangerous Function

Safety is critical when handling power tools. 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. If potentially-dangerous functions are not used properly, then things can get real messy real quick.

No alerts in this category.

(19)Use of a Broken or Risky Cryptographic Algorithm

If you are handling sensitive data or you need to protect a communication channel, you may be using cryptography to prevent attackers from reading it. 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. Cryptography is just plain hard. If brilliant mathematicians and computer scientists worldwide can't get it right (and they're always breaking their own stuff), then neither can you. You might think you created a brand-new algorithm that nobody will figure out, but it's more likely that you're reinventing a wheel that falls off just before the parade is about to start.

No alerts in this category.

(20)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. Any number of problems could produce the incorrect calculation, but when all is said and done, you're going to run head-first into the dreaded buffer overflow.

No alerts in this category.

(21)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. Without some kind of protection against brute force techniques, the attack will eventually succeed. You don't have to be advanced to be persistent.

No alerts in this category.

(22)URL Redirection to Untrusted Site ('Open Redirect')

While much of the power of the World Wide Web is in sharing and following links between web sites, typically there is an assumption that a user should be able to click on a link or perform some other action before being sent to a different web 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. This may be another of those features that are "just the way the web works," but if left unchecked, it could be useful to attackers in a couple important ways. First, the victim could be autoamtically 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.

No alerts in this category.

(23)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 - sometimes, even, to execute code.

No alerts in this category.

(24)Integer Overflow or Wraparound

In the real world, 255+1=256. But to a computer program, sometimes 255+1=0, or 0-1=65535, or maybe 40,000+40,000=14464. You don't have to be a math whiz to smell something fishy. Actually, this kind of behavior has been going on for decades, and there's a perfectly rational and incredibly boring explanation. Ultimately, it's buried deep in the DNA of computers, who can't count to infinity even if it sometimes feels like they take that long to complete an important task. When programmers forget that computers don't do math like people, bad things ensue - anywhere from crashes, faulty price calculations, infinite loops, and execution of code.

No alerts in this category.

(25)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. If you don't add a little salt to your hash, then the health of your application is in danger.

No alerts in this category.

Affected Items: A Detailed Report

This section provides full details of the types of vulnerabilities found according to individual affected items.

Scanned items (coverage report)

http://sitalitbang.semarangkab.go.id/
http://sitalitbang.semarangkab.go.id/icons
http://sitalitbang.semarangkab.go.id/manifest.json