Running Title: Investigating SANS/CWE Top 25 Programming Errors.

# Investigating the SANS/CWE Top 25 Most Dangerous Programming Errors List

Fred Williams

East Carolina University

ICTN 6870

# Abstract

On January 12, 2009, experts from more than 30 cyber security organizations jointly released a consensus list of the top 25 most dangerous programming errors (http://www.sans.org/top25errors/). This list attempts to boil down the more than 700 possible causes of software security issues to the ones that are so prevalent and severe that no software should be released to customers without evidence that measures were taken to ensure the software does not contain any of these errors. The Top 25 errors were further broken down into 3 categories: Insecure Interaction between Components that contains 9 errors, Risky Resource Management which contains 9 errors and Porous Defenses has the final 7 errors.

Maybe not surprisingly, many programmers and software testers are not properly trained to recognize, isolate and resolve these issues. Some believe that it is not more that software developers are not educated in secure programming techniques but more that C level executive will not build time into the project schedule to apply the principles. To solve the education part and promote safer coding and testing practices, this paper will attempt to examine the big hitters of the list: the errors that are most prevalent, the most frequent and the ones that factor most into other, more complex errors. These heavy hitters are not simply in the first category or ranked in the top 10 but spread among all 3 categories. By explaining each big hitter and providing code samples, this paper will reach the goal of providing education to software developers, testers and project management that will lead to more secure software for the most sensitive customer facing web applications. Now, the only issue for employees to solve is to get buy- in from senior level management.

## Introduction

An old software proverb states: "The first 90% of the work takes 10% of the time and the other 10% takes 90% of the time". Security, sadly, is relegated to the latter and is not at the forefront of the development cycle. Sometimes, security is not gotten around to implement at all. More concerning is the situations where security is deemed not important enough to build into the application. As stated in the abstract, a lot of times it is not the fact that development staff do not know the solutions for thwarting many of the common malicious attacks that happen today, it is more that management will not allocate resource in the budget. Therefore, many software developers are not well versed in techniques required to ensure that the systems they build are protected against the myriad of attacks proliferating the Internet today. Even as the world's economy continues to deteriorate only means that these malicious attacks will rise exponentially in order to provide monetary gains to criminal gangs trying to exploit lax security. Don't think perimeter defense is enough to combat most types of attacks that occur the most frequently and remember that "…firewalls cannot protect against all vulnerabilities such as service and application." (Dasgupta, Ferebee, May 2008). The SANS Institute and the MITRE organization compiled a list of the common vulnerabilities that can occur in software as a way to raise awareness of the seriousness of ignoring secure coding techniques. The main goal was to stop these attacks at the source by educating software developers and testers on how to eliminate all-too common mistakes before software ships.

This paper will first explore the important organizations behind the Top 25 Most Dangerous Programming Errors list and help explain why they released this list in the first place. Next will come a brief listing of all of the errors in the list with examples of what I consider the 'heavy hitters' – the errors that have a wide prevalence and higher frequency of occurrence than some of the others. The idea of focusing on the heavy hitters is that if your team only has a limited amount of resources, by analyzing and fixing these, you can get more bang for your buck. By providing examples, targeted audiences can use these simple examples to search for more detailed information that relates to their specific problems. Finally, a series of lessons learned will be presented as an attempt to pull the covers back from the

contents of the list and boil down to a few takeaways that developers and testers can use to aid in creating more secure web applications.

## Most important organization behind the SANS/CWE Top 25 Programming Error List

The SANS (SysAdm, Audit, Network, Security) Institute was founded in 1989 as a cooperative research organization that provides for free the largest collection of material about various aspects of information security. They also claim the largest source of information security certifications as well. SANS is behind the Internet Storm Center, an early warning system for Internet service providers around the Internet that attempts to detect, isolate and resolve malicious attacks as they occur. The ISC collects data from over half a million IP addresses in 50 countries to determine where the most active malicious content is thriving. According to Wikipedia, "its website cites a figure of over 20 million 'intrustion detection log entries' per day. It continues to provide analysis and alerts of security threats to the Internet security." (Wikipedia, 2009).

The MITRE organization is a think tank that operates three federally funded research facilities where engineers explore ideas and theories related to information security. MITRE also has an independent branch that still explores technology including security, but is not related to the government departments. The Common Weakness Enumeration (CWE) is a list of software weaknesses maintained by MITRE that "provides a unified, measurable set of software weaknesses that is enabling more effective discussions, description, selection and use of software security tools and services that can find these weaknesses in source code and operational systems as well as better understanding and management of software weaknesses related to architecture and design" (MITRE, 2009, CWE Overview). The original CWE list is currently versioned at CWE Version 1.3 that brings the total number at 732. The MITRE organization, in collaboration with the SANS Institute, the Department of Homeland Security and more than 40 other top security experts in the field examined the SANS Top 20 attack vectors and MITRE's own CWE list to develop the finalized top 25. To provide a context in which to determine what errors were most dangerous, a threat model was designed to document the average type of attacker. Part of the threat

model was to categorize the skills and determination of a typical attacker as well as a primary means of motivation. This threat model assisted the experts to determine which errors should be included in the top 25.

## Why a Top 25 Programming Errors List?

Why do we need yet another list? It is true there have been more than a few top errors list geared towards information security over the years. Matter of fact, the MITRE CWE website provides a complete list of taxonomies that served as classifications and sources from a multitude of historical lists and provided input to the top 25 list. On this site you will see several of the most recognizable lists. Michael Howard's "19 Deadly sins of software security" claims you can secure your software by eliminating code vulnerabilities from the start. Howard, who teaches Microsoft employees on how to write secure code, covers many of the issues that are on the SANS/CWE list like SQL Injection and race conditions.

Steve McConnell, author of many award winning books and former editor-in-chief of IEEE Software, came up with a list he called Classic Mistakes Enumerated. McConnell found that certain mistakes were committed over and over again, in the same fashion and by the same employees in such a predictable pattern that they deserve to be called Classic Mistakes. He even created a case study that follows a development team at a fictitious company as the management and employees bungle a software schedule while committing the same mistakes as in McConnell's list. The cast study is a very effective way of demonstrating a real world scenario that people can relate to.

The Open Web Application Security Project (OWASP) group has perhaps one of the most informative, well organized and widely respected software error lists in existence today. The OWASP Top 10 list has been the basis for many educational lessons and code review tools coverage analysis and may be the defacto standard list up until now.

## The Error List.

Let's start digging into the contents of the list by examining the 3 categories the experts assigned each of the 25 errors. These categories attempt to logically relate each error to one another. The categories and explanation of each are:

1) **Insecure interaction among components –** This category examines 9 ways different software components interact, call, and use other software components using insecure methods.
2) **Risky resource management –** This category contains 9 errors that explain how critical system resources are not properly managed. These resources can include configuration files, memory, software downloads and source code.
3) **Porous defenses –** This category contains 7 errors that show how proper security defenses are misused, abused or sometimes simply ignored.

The first category – insecure interaction among components - contains what I call many of the big hitters. The big hitters are the errors that either factor into many other errors or are of high prevalence and frequency.

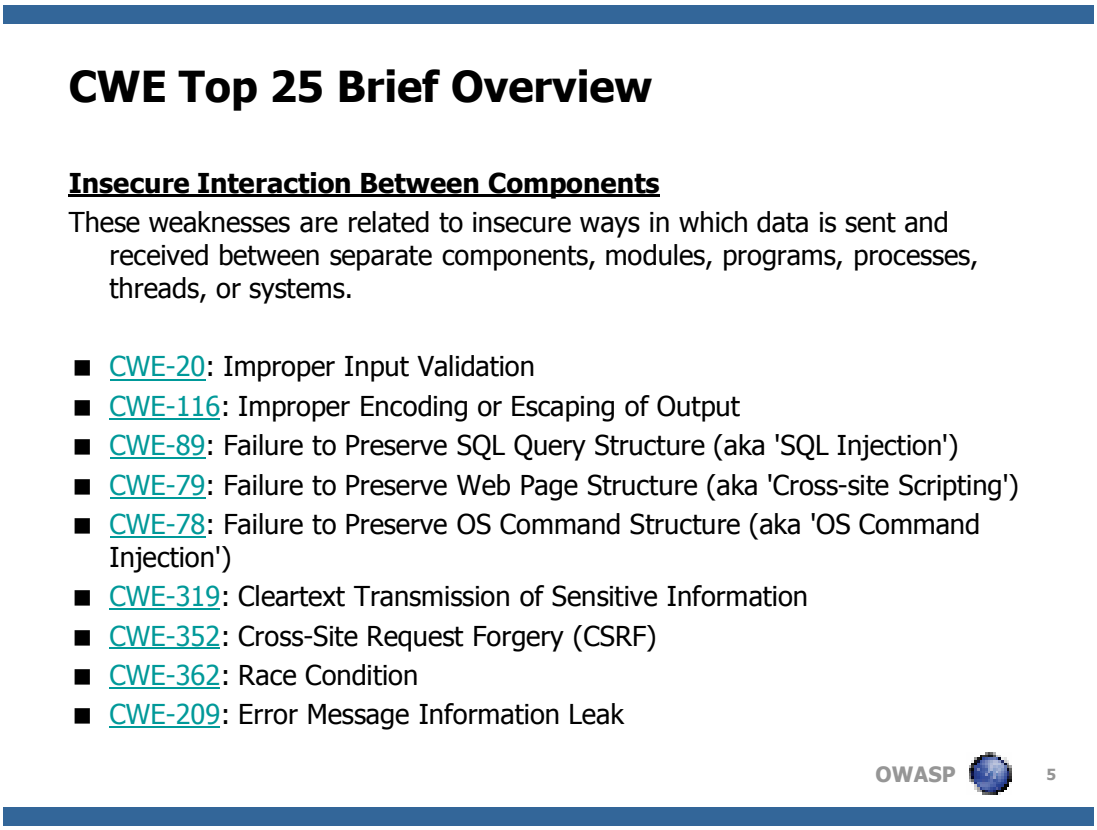


Figure 1 - Top 25 Programming Error List First Category

There are still a few big hitters in other sections and I don't think the authors intentionally ranked the 25 errors in order of importance but the idea here is to look at the first category when determining a basis for the list. Matter of fact, the very first two errors that cover proper input validation and proper encoding factor into more than half of the entire list and should be the first thing a team should focus on. Study **CWE-20 Improper input validation** and **CWE-116 Improper encoding or escaping of output** to learn more about the factors and solutions to lessen the impact of attacks. Then, when you get down later in the list to injection and cross site scripting attacks, you will see that it all comes back to the first two. Take an in-depth look at your application and try to identify each possible source of input especially for web applications and services. These types of software applications can receive input from humans entered via web forms, it can come in the form of XML in SOAP message payloads, and finally HTTP requests parameters in an URL string. The OWASP group provides a very helpful cheat sheet in determining all possible inputs to your software called the OWASP Code Review guide. Jeff Williams, OWASP Chair,

says that "The Development Guide shows your project how to architect and build a secure application and the Code Review guide tells you how to verify the security of your application's source code." [Williams 2007].

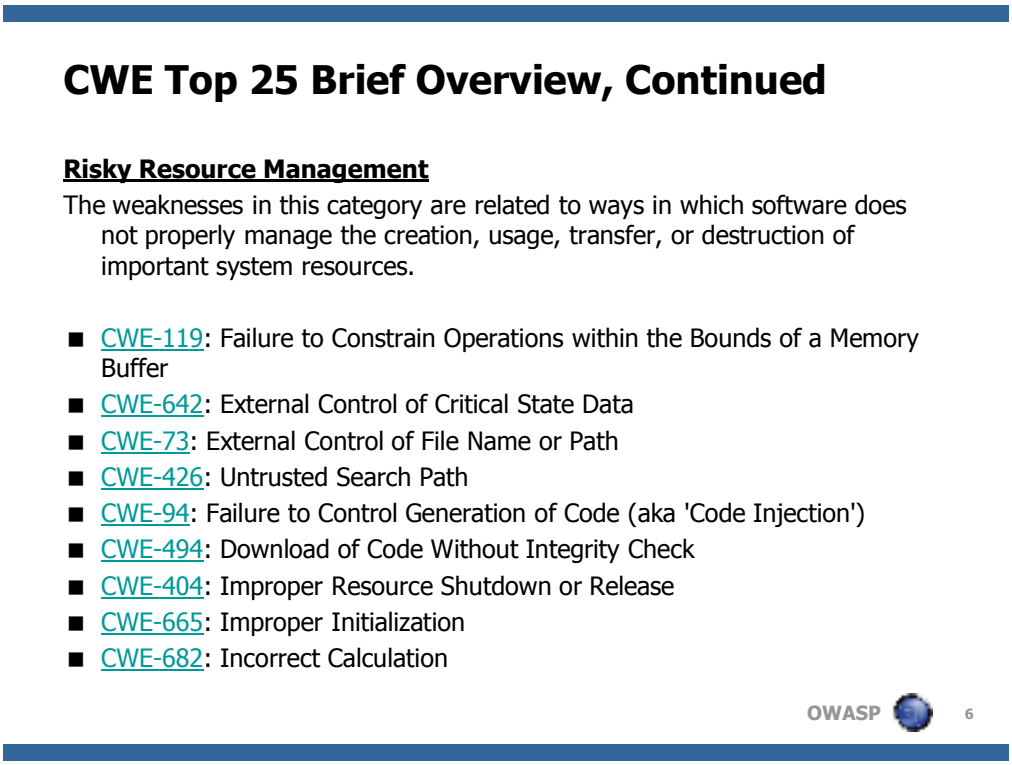


Figure 2 - Top 25 Programming Error List Second Category

The second category of errors is listed above. Now, I wouldn't call any of these errors miniscule by any means and there are some very serious vulnerabilities listed in this category. However, if time is of any consequence in your team's plans, please move on to tackling the big hitters in the third category. Matter of fact, our two biggest hitters, input validation and proper encoding, factor in at least three of the nine listed in this category. If your team must focus on any of these nine, I would recommend researching **CWE-404 Improper resource shutdown or release** because serious problems like database connection pool exhaustion and cookie mismanagement are two problems that can be addressed in this category. If your project does not focus on CWE-404, attacks such as Denial of Service could affect your applications.

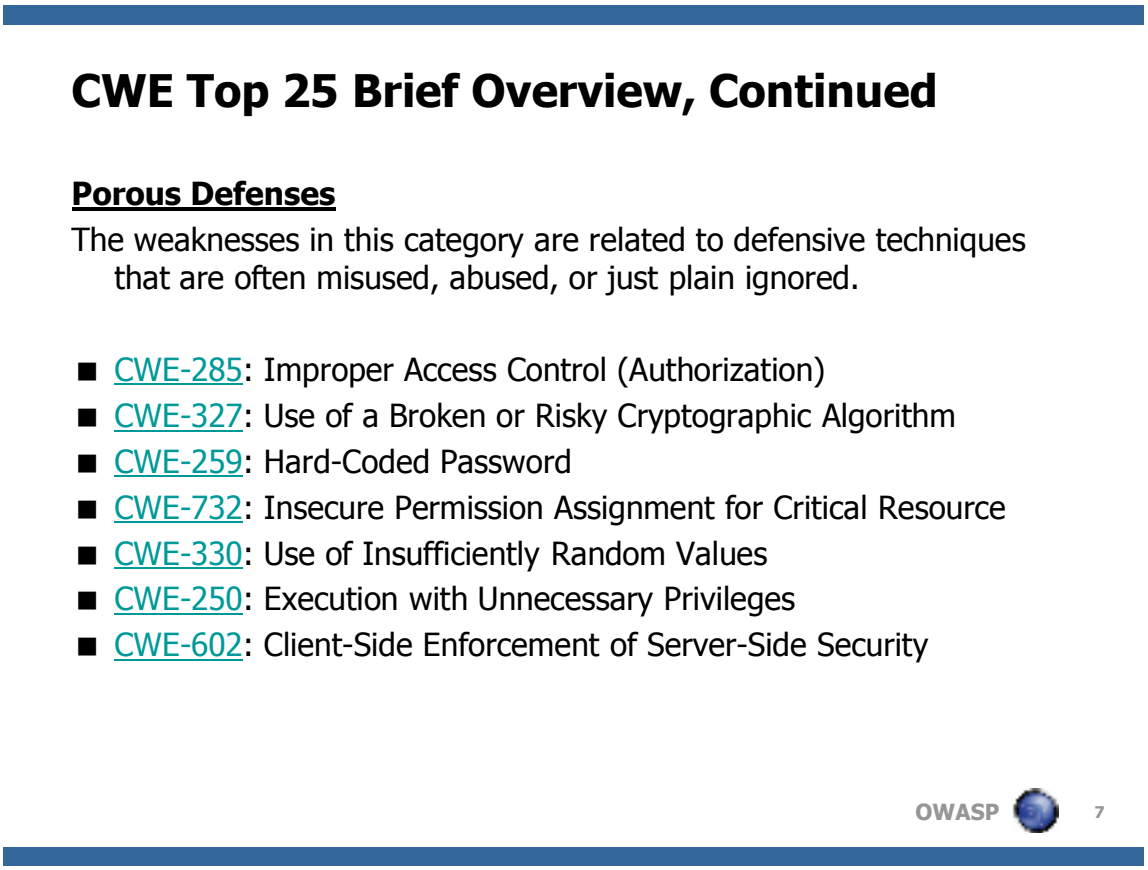


Figure 3 - Top 25 Programming Errors Third Category

Finally, our last category lists errors that can occur when applications do not take the time to shore up proper defenses or ignore defenses entirely. Out of these seven errors I would consider **CWE-250 Execution of unnecessary privileges** the most important. The practice of least privileges is number two in this paper's list of 5 recommendations to take away for future considerations since other errors most of the time do not cause as much damage if a user has limited permissions rather than super user type of privileges. If you use a role based authentication scheme make sure to apply the security checks on sensitive processes to only allow higher authority users. Then make sure to apply the checks on the server side as frequently as possible. Frameworks such as the Spring Framework offer authentication and authorization schemes that work by configuring providers and filters to allow specific roles access to URL patterns and web services.

## Lessons learned from the Top 25 Most Dangerous Programming Error List.

If we pull back the covers of this list and try to focus on overall messages that can be taken away, what would we find? Even though not specifically spelled out, a few major touch points can be inferred. Here are 5 important lessons learned:

1) **Start by identifying all possible entry points to your application and white listing inputs.**
2) **Employ the practice of running with least privileges.**
3) **Pay special attention to SQL Injection attacks.**
4) **Use a mature framework as a basis for development.**
5) **Code Reviews and testing are very important.**

Let's explore each of these touch points with some specific recommendations for any information technology department that produces customer facing software applications.

**Start by identifying all possible entry points to your application and white list inputs.**

Probably the most important thing that a software development team can do is to identify all possible inputs to their applications and protect those inputs against malicious attacks. Follow CWE-20 and CWE-116 by applying proper input validation and proper output escaping/encoding. While there are a huge number of vectors for XSS and injection attacks, following a few simple input validations can completely defend against these serious attacks. Be a good citizen - as applications become more interconnected with other outside processes in the form of web services, the likelihood of a buried attack being executed downstream increases exponentially. You can use the OWASP code review checklist as mentioned earlier to assist in code reviewing for attack weaknesses.

White listing can be a little more difficult. What is white listing and how does it compare with black listing? White lists are defined as trusted sources of content while black lists contain content that are known to be untrustworthy. For example, what if a Voice-over-IP application is being bombarded with voice spam similar to how telephones suffer from telephone solicitors? "A white list module compares

the caller's identity to a database of stored trusted identities. A black list module compares the caller's identity to database entities deemed not to be trusted. " (Mathieu, Niccolini, Sisalem, 2008, p. 56). This paper's recommendation is to choose white listing over black listing. The reason is that if an application is comparing inputs against a black list of known malicious sites, what happens when a new site is added or a new way of bypassing existing checks are used? The application must update the black list or potentially be compromised by an attacker successfully exploiting the gap. If you rely on white listing to only accept known good inputs, then all other inputs will be considered malicious and the program is more likely to remain secure.

The most common method that programmers employ to compare inputs against white lists is usually by writing regular expressions. A regular expression is a pattern composed of a sequence of characters that contains only known accepted characters. For example, if you have a web form with a field for entering a person's name, then you may write a regular expression that only accepts A – Z characters with mixed case rules. OWASP again comes to the rescue here by providing the OWASP Enterprise Security API (ESAPI) Toolkit http://www.owasp.org/index.php/ESAPI#tab=About . The OWASP ESAPI can assist the development team by providing utility method that apply proper input validations and proper encoding routines.

Another potentially more fatal flaw could be the issue of multiple alternate encodings of input supplied to an application. These flaws could occur because data that are encoded in Unicode, UTF-8, or HTML or some other structured encoding schemes allow for multiple representations of characters. For example, the built in HTML parsing utilities in modern web browsers will consider these two strings exactly the same:

- `&#60;&#115;&#99;&#114;&#105;&#112;&#116;&#62`
- `<SCRIPT>`

So, if the developer creates a white list for detecting potential injection attacks but does not account for the HTML encoding scheme as illustrated above, then the software is still open to this type of attack. Given the possibility of multiple encodings, the developer must first translate all inputs to a single, standard, minimal representation. This translation is referred to as canonicalization and it allows for inputs to be compared with a single representation.

**Employ the practice of least privileges for users.**

Some security professionals may argue that this remediation could be more important that proper input validation. The idea here is just simply to give users just enough permission that they need in order to do their job and not a single permission more. The consequences of most flaws contained in the Top 25 Most Dangerous Programming error list usually results in a privilege escalation, an important goal of most attackers. Higher levels of privileges can allow attackers the ability to change administrative account information, add bogus user accounts to a database and allow a future uses of the system with the same greater level of system privileges. The practice of least privileges suggests that granting a super user or administrative level of privileges should be granted as rarely and as briefly as possible. In addition, a good idea is to always restrict ownership of files and directories available to the software which is a common deficiency of today's web application. In most typical default web server settings, it is common for the web server to have full control of all files and directories under the web root folder. Therefore if an attacker is able to gain entry into the system through the web server, then the attacker will enjoy full control. Inspect your web server settings and modularize web applications and reduce user privileges to that of another special lower permission user. Any subsequent attack is then much less significant.

**Use a mature framework as a basis for development.**

For any new projects that your team is planning for future development consider a mature open source framework as a basis for your new development. It isn't an easy undertaking to retrofit existing code base

with a framework especially if the number of lines of code exceeds 100,000 lines. However, new projects that are still in the design phase especially if the architecture is not stable a framework like the Spring framework will offer many advantages to a development team.

Spring offers solutions to many of the issues in the Top 25 error list: an authentication and authorization mechanism through Spring Security, a validation framework that servers both the client and server side and built in libraries that handle sensitive boilerplate codes like opening and closing critical system resources. A really nice feature of modern frameworks is the ability to automate software testing procedures that are so important for finding security flaws. Spring has mechanisms to hook into the popular JUnit testing framework and provides helper classes that wrap and extend JUnit that make the already powerful JUnit automated testing API much more powerful.

**Pay special attention to SQL Injection attacks.**

SQL Injection attacks are just one example of what is classified as code injection attacks. "The term injection attack refers to a wide variety of program flaws related to invalid handling of input data". (Stallings, Brown, 2008). However, out of all the errors in this category SQL injection attacks are the errors that occur most frequently and are increasingly becoming more severe. According to the Breach Security, a leader in web application security, attackers are using SQL injection more today for financial gains and stealing personal information, which is easily traded for money and has become the top virtual commodity. Breach's Web Hacking Incident Database (WHID) report stated "Marking a major event for the web application security landscape, the report found that SQL injection attacks planting malware on target web sites was the number one security attack for online criminals last year". (CIOL, 2008) The logical thinking would say that if a project focuses on preventing SQL injection attacks then a large part of all malicious activity would be defended against. What is the most efficient way to deal with this type of weakness? The *IEEE Transactions on Software Engineering* Journal mentions that "..if user input is not properly validated, attackers may be able to change the developer's intended SQL command by

inserting new SQL keywords or operators through specially crafted input strings. “. (Halfond, Orso, Manolios, Jan.–Feb 2008). There are a few of these listed in the Top 25 list and the common solution among these always seems to point back to sanitizing software inputs.

**Code Reviews and testing are very important.**

Catching software defects and security flaws early in the design and coding phase are much more inexpensive that later in the process. Add to that the incalculable cost of customer deployment delays or security breaches and that could have negative consequences to an organization’s reputation. Today’s software professionals use mature tools when developing software and if you choose integrated development environments the team also gets built in testing tools. Several popular IDEs for developing Java, Ruby and Spring applications are Eclipse and Netbeans. One very nice tool that assists in checking code for security issues is CodePro. A recommendation from this paper would be to use the OWASP Code Review Guide in conjunction with a tool like CodePro to assure expert advice from experienced developers. Here is a screen shot for setting preferences that will tell CodePro which vulnerabilities to scan for:

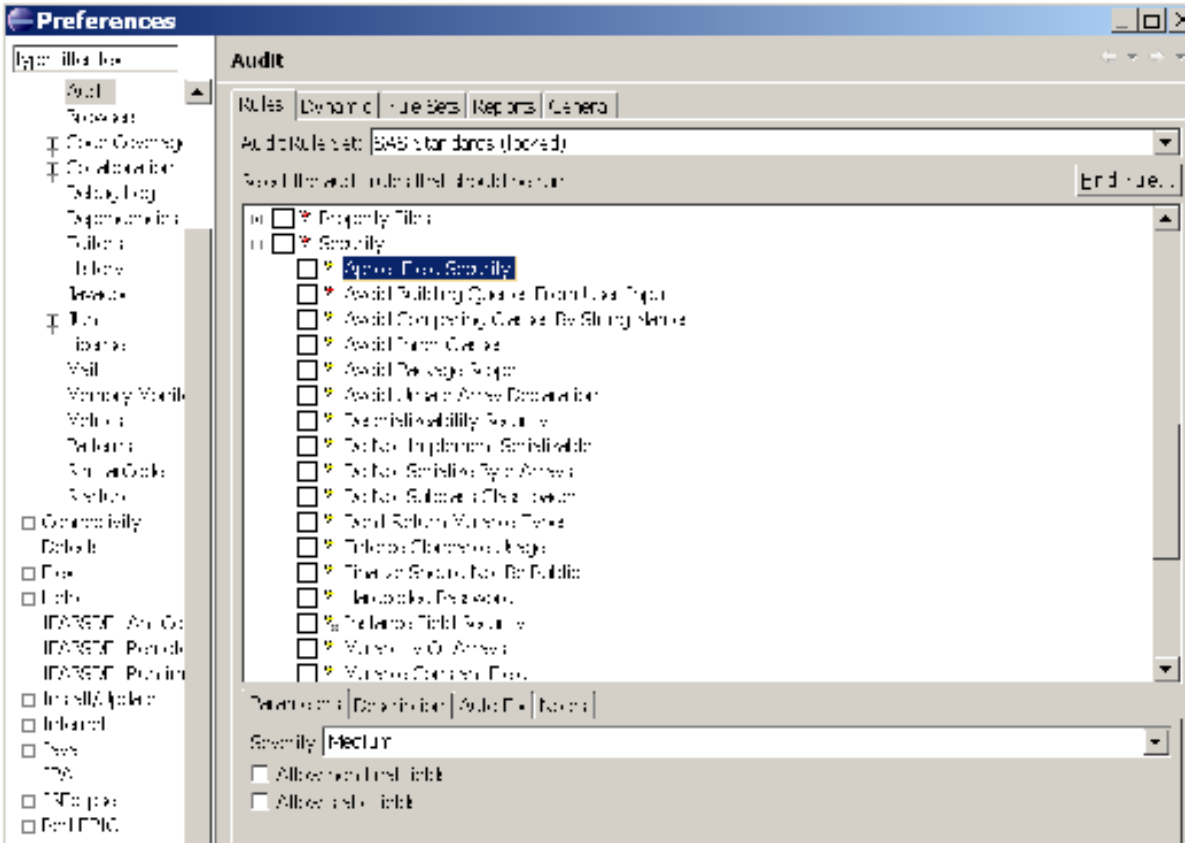


Figure 4 - CodePro Analytix Audit Rules configuration screen in Eclipse.

For example, remember the first recommendation of this paper revolving around sanitizing user inputs, according to CodePro's website, CodePro has a built in security test that tries to identify tainted user inputs. "The classic example of [this] method is the `executeQuery(String)` method defined for SQL statements. If the user can control the content of the string they may be able to get far more information than they ought to be able to get…" (CodePro User's Guide, 2009). Let a tool like CodePro identify automatically as much as possible to reduce the workloads on the developer.

Conducting regularly scheduled code reviews for the team add that extra needed element of human manual inspection of code for weaknesses that automated testing tools cannot. In addition, code reviews by senior members of a team on less experienced developers promote knowledge sharing that helps educate all members of the staff.

A recommendation from this paper includes scheduling regular team code reviews, use modern integrated development environments like Eclipse and develop reusable routinely processes automated test suites.

# Conclusion

It has become almost a duty for organizations to be good citizens on the Internet by striving to develop the utmost in quality and secure software as possible. Software attacks can result in catastrophic consequences to not only the targeted applications but the users and ultimately its customers if these attacks result in success. The image and reputation of the company is outright affected but thieves can also steal the identities and ruin the lives of the customers who inadvertently become the victims of such attacks. Perimeter defenses such as routers and firewalls are not enough to thwart many of today's common web application attacks such as SQL injection and XSS. How can developers, quality assurance staff and project managers learn defensive programming techniques? Education and awareness are probably the top two solutions that can be applied towards helping IT staff learn and implement secure coding techniques. It will teach quality assurance staff what to test and pay attention to when it comes to common vulnerabilities such as code injection attacks. Top programming error lists such as the Top 25 Most Dangerous programming error list jointly released by the SANS Institute and the MITRE Organization can also help. The SANS Institute and the MITRE organization did the hard part for us by narrowing the more than 700 possible Common Weakness Enumeration errors to the ones that are most prevalent, occurs the most frequently and wreaks the most havoc. This gives a development team a basis for creating automated tests and testing tools that assist in this effort. If we examine this list a little more closely and see common themes to each of the 25 errors we can pick out a few lessons to take away and apply to future development activities. Working towards identifying and sanitizing all user supplied inputs to applications to accept only known, appropriate data can probably reduce vulnerabilities by at least 50%. The reason is that improper input validation and improper encoding of outputs, the first two items in the Top 25 error list, factor into at least half of the errors in the list. By applying techniques mentioned by these two items as well as the other 23 issues can aid a development team by creating more secure software and reduce downstream effects to companies and customers alike.

# References

MITRE Organization website. (2009, March 9). In *Common Weakness Enumeration Overview.* Retrieved March 17, 2009, from http://cwe.mitre.org/

Howard, Michael. (2003). *Writing Secure Code.* Redmond, Washington: Microsoft Press. Retrieved March 19, 2009 from O'Reilly Safari Books Online http://safari.oreilly.com/.

CodePro User's Guide. (2009). "A Description of the Tainted User Input Rules". *In CodePro User's Guide.* Retrieved March 23, 2009 from Instantiations website - http://download.instantiations.com/CodeProDoc/continuous/latest/docs/doc/features/audit/tainted_user_input_description.html.

Williams, Jeff (2007, October 17). "Foreword by Jeff Wiliams, OWASP Chair". *In OWASP Code Review Guide v1.1.* Retrieved April 4, 2009 from OWASP website - http://www.owasp.org/index.php/Category:OWASP_Code_Review_Project .

* Mathieu, B., Niccolini, S., Sisalem, D. (2008). "SDRS: A Voice-over-IP Spam Detection and Reaction System". *IEEE Security & Privacy*, *Volume 6, Issue 6, November-December,* p.56. Retrieved April 14, 2009 from IEEE Xplore database http://ieeexplore.ieee.org/Xplore/guesthome.jsp.

* Dasgupta, D., Ferebee, D. (2008). "Enhancing Computer Security with Smart Technology [Book Review]". *IEEE Computational Intelligence Magazine*, *Volume 3, Issue 2, May 2008,* p.70 - 71. Retrieved April 15, 2009 from IEEE Xplore database http://ieeexplore.ieee.org/Xplore/guesthome.jsp.

* Halfond, W.G.J., Orso, A., Manolios, P. (2008). "WASP: Protecting Web Applications Using Positive Tainting and Syntax-Aware Evaluation". *IEEE Transactions on Software Engineering*, *Volume 34, Issue 1, January-February,* p.22-29. Retrieved April 14, 2009 from IEEE Xplore database http://ieeexplore.ieee.org/Xplore/guesthome.jsp.

Cybermedia India Online Ltd. (2009, February 26). "Hackers are becoming more savvy: WHID" on *Cybermedia India Online Ltd.* Retrieved April 19, 2009, from http://www.ciol.com/Developer/Databases/Feature/Hackers-are-becoming-more-savvy-WHID/26209116530/0/

Stallings, W., Brown, L. (2008). *Computer Security Principles and Practice, (p.394).* Upper Saddle River, New Jersey: Pearson Education, Inc., Pearson Prentice Hall.

Internet Storm Center. (2009, April 19). In *Wikipedia, the free encyclopedia.* Retrieved April 19, 2009, from http://en.wikipedia.org/wiki/Internet_Storm_Center.