veracode.com

The Language of AppSec

By Brian Pitta

5-6 minutes

Everyone has weird language issues they just can't get right – mine is ordering at Starbucks. If the store doesn't have sizes on display that I can awkwardly point to, I end up panicking, ordering a "tall," and walking away disappointed with my small coffee. Starbucks and I just can't speak the same language (yes, it's my fault).

This problem of speaking a different language is not unique to my afternoon pick-me-up. When I hear "language" in regards to application security, my mind typically goes to Java, C#, PHP or any of the other common development languages. However, there is a whole other language that is just as important: the language used to communicate ***about*** AppSec.

Application security can be complicated. Making sure you are speaking the same language with your team (internal and external) is one of the best ways to simplify it. After working with hundreds of organizations in different industries, I've come up with the top three AppSec language confusions I commonly hear, and it turns out they are all related.

1. Flaw vs. Vulnerability

At first glance, they might sound close enough, but they are quite different when you dive into it. A flaw is a weakness in an application that needs to be investigated. A vulnerability is a flaw that has a proven exploit.

Static analysis models an application's data and control flows to comprehensively identify any flaws in the code. When a static scan identifies a flaw, it doesn't always mean it's exploitable; it means it needs to be reviewed. If you determine that the flaw could be exploited, you remediate the code to remove the risk and prevent it from becoming a vulnerability.

Dynamic analysis and manual penetration testing find vulnerabilities in applications in a runtime environment and can capture the proof of exploit.

2. CVE vs. CWE

These are both industry standards for communicating findings from some sort of assessment.

The Common Vulnerabilities and Exposures (CVEs) is a naming convention for documenting vulnerabilities discovered in software. For example,

- ***CVE-2014-1904*** is a XSS vulnerability in the Java Spring MVC framework.
- ***CVE-2015-2213*** is a SQL injection vulnerability in WordPress.

The Common Weakness Enumeration (CWE) is the categorization of software weaknesses. For example,

- ***CWE-79: Improper Neutralization of Input During Web Page***

Generation is the weakness (or flaw) in the code of the Spring MVC framework that caused CVE-2014-1904

- ***CWE-89: Improper Sanitization of Special Elements used in an SQL Command*** is the weakness (or flaw) in the code of WordPress that caused CVE-2015-2213.

I think about it in terms of cause and effect: a flaw in the code (CWE) will cause a vulnerability or exposure in the software (CVE). A good AppSec program can help find these flaws before they become vulnerabilities. Veracode's static analysis, dynamic analysis, and manual penetration testing identifies CWEs in your software, and our Software Composition Analysis identifies CVEs in your third-party and open-source components.

3. Mitigate vs. Remediate

These two words can be used interchangeably, but in Veracode lingo, we've standardized on two distinct meanings. When I ***mitigate*** a flaw, I am documenting a compensating control that I believe adequately addresses the risk associated with it. When I ***remediate*** the flaw, I'm changing the code to address the risk.

An example best illustrates this. Let's look at the scenario where a static scan finds a SQL injection flaw in my application. The scanner identified a location where I am taking data from outside the application and using it to dynamically construct a query without sanitizing it. If I review this flaw and determine the tainted data source is a database that has strict controls and doesn't take user input, I might mark this flaw as **Mitigated by Design** in the Veracode platform.

On the other side, if I look at this flaw and determine I'm taking data from an HTTP request and using it to construct the query, then this flaw could be exploited. In this case, I'm going to **remediate** the code by sanitizing the data or parameterizing the query to remove any SQL injection risk.

We'd love to hear some of the common points of confusion for your team. What AppSec language do you use that helps clarify communication?

Brian Pitta is a Senior Solutions Architect at Veracode. He works with Veracode's customers to identify the best approach to securing their applications and helps establish a successful AppSec program. He started his career as an Environmental Engineer and made the move to security once he saw some funky behavior on the fantasy golf application he developed. Outside of work, you can find him on a ping-pong table or disc golf course.