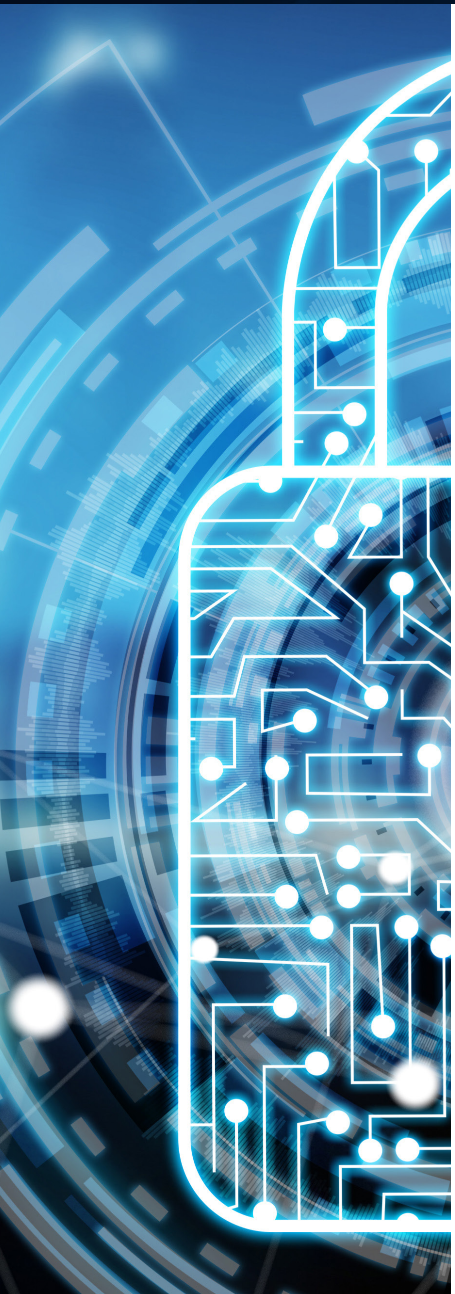
Embedded Cybersecurity Through Secure Coding Standards CWE and CERT

INTRODUCTION

Cybersecurity is an increasingly important topic today, as more and more software is connected and can be exploited to make software and devices act in ways other than originally intended.

Most organizations are coding at least some security testing, usually in the form of black-box testing, penetration testing, or red teams. This is important, but an engineering approach is to build better, more secure software in the first place. That's where coding standards come in. While some standards are related to finding security defects through techniques like flow analysis and taint analysis, other standards help identify code that could be compromised, or coding methods that prevent compromise. In this paper, we'll take a look at two common cybersecurity coding standards, CWE and CERT.



CWE

The CWE (Common Weakness Enumeration) is a list of problems that can occur in code and lead to exploitable security issues. CWE complements CVE (Common Vulnerabilities and Exposures) by describing the code that lies behind software vulnerabilities. CWE has been built by many contributors from the software security community. It's managed by Mitre and sponsored by the US Computer Emergency Readiness Team (US-CERT) and the US Department of Homeland Security (DHS).

The main goal of the CWE initiative is to stop vulnerabilities at the source by educating software acquirers, architects, designers, and programmers on how to eliminate the most common mistakes before software is delivered.

CWE FAQ

This list of over 800 types of problems that can occur in code is comprehensive, but can be overwhelming, so there is a "Top 25" version that is based on issues that are mostly commonly happening and have some kind of nasty outcome as well. It's a simple way to get started by focusing on things that matter the most and are most likely to happen. Of course, it's easy to fall into a trap of complying with the Top 25 and then thinking you're finished. The Top 25 is just a starting point on your secure software journey.

CERT

CERT is a project of the Software Engineering Institute (SEI) at Carnegie Mellon University (CMU). They have created the CERT secure coding standards for a variety of languages, with a focus on hardening your code by avoiding coding constructs that are more susceptible to security problems.

PARASOFT®
Automated Software Testing

"we research and develop solutions for identifying and preventing security flaws during development, where it is much more cost effective than in the test phase or post-deployment"

SEI web site

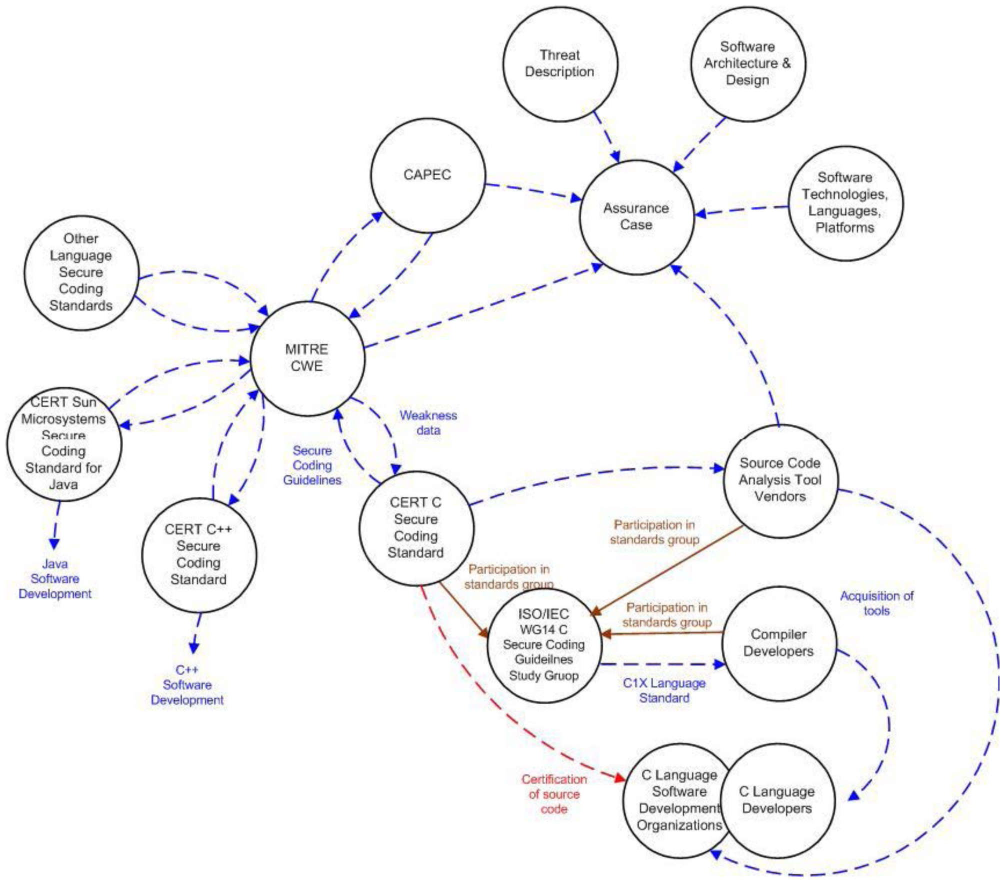
CWE OR CERT

Both of these standards are based on research that shows that simply testing software isn't getting the job done. You can't test security into an application any more than you can test quality into an application – you have to build secure software from the ground up. They also acknowledge that many security vulnerabilities are simply exploiting underlying quality flaws.[1]

A common question regarding CWE and CERT is, "Which should I use? Is one better than the other?" These two security projects complement each other, and each has a bit different charter. CWE is a list of software weaknesses that may be used to exploit a software system. There is a detailed statement about this that you can find on both of them on the SEI web site.

"The Common Weakness Enumeration (CWE) is a unified, measurable set of software weaknesses that enables the effective discussion, description, selection, and use of software security tools and services that can find these weaknesses in source code and operational systems."

SEI web site


Threat Description
Software Architecture & Design
CAPEC
Assurance Case
Software Technologies, Languages, Platforms
Other Language Secure Coding Standards
MITRE CWE
CERT Sun Microsystems Secure Coding Standard for Java
Weakness data
Secure Coding Guidelines
Source Code Analysis Tool Vendors
CERT C Secure Coding Standard
CERT C++ Secure Coding Standard
Java Software Development
Participation in standards group
Participation in standards group
ISO/IEC WG14 C Secure Coding Guideilnes Study Gruop
Participation in standards group
Compiler Developers
Acquisition of tools
C1X Language Standard
C++ Software Development
Certification of source code
C Language Software Development Organizations
C Language Developers

CERT-CWE Landscape[2]

[1] http://swreflections.blogspot.com/2015/01/if-you-got-bugs-youll-get-pwned.html
[2] https://resources.sei.cmu.edu/asset_files/WhitePaper/2010_019_001_50405.pdf

CERT is a coding standard designed to help you avoid weaknesses by avoiding error-prone constructs and patterns.

> *"The CWE and the CERT secure coding standards perform separate but mutually supportive roles. Simply stated, the CWE provides a comprehensive repository of know weaknesses, while CERT secure coding standards identify insecure coding constructs that, if present in code, could expose a weakness or vulnerability in the software."*
>
> *SEI web site*

When applied, software security incidents like breaches exploit weaknesses in software (CWE) and proper guidelines and best practices (CERT) tell you have to avoid having such weakness in your system.

> *"Guidelines in the CERT C Secure Coding Standard are cross-referenced with CWE entries. These cross-references are only created for guidelines which, if violated, directly contribute to the referenced weakness. Similar mappings will be created for other CERT coding standards once they are completed"*
>
> *SEI web site*

FITTING IT ALL TOGETHER

It's easy to get overwhelmed by all these acronyms. Let's try putting it together now in a useful way. For software, the CVE is a unique ID that describes a known software vulnerability that can be exploited in the real world. When analyzed, a particular CVE problem has some root causes in the code behind it – those are the CWEs. For a denial-of-service issue (DDoS or DoS) we may find code that is subject to buffer overflow. The DoS gets a CVE ID. The code that overflows gets a CWE ID. CERT rules tell you how to avoid the overflow in the first place by coding differently.

A real-world way to look at it is to think about CVE as a traffic accident. CWE is the cause or causes behind the accidents, i.e. poor visibility at an intersection, bad brakes, driver distractions, following too closely, etc. CERT is avoidance systems, like putting mirrors on blind corners, having 3rd brake lights in cars, ABS brakes, etc.

This is important to emphasize because people tend to focus on the testing part of security ("Where is a breachable point?"). This is a worthy exercise, but should be followed by, "How could I have prevented that breach in the first place?" as opposed to, "Let me patch that hole." CERT and CWE will help you do this when applied properly.

This can be a challenge because software developers have been conditioned to think about security tools finding specific vulnerabilities for them, even with exploit data. When they get a tool message that says "buffer overflow can happen – here's a specific piece of data that can make it happen" they are happy. When a tool tells them "This kind of code is susceptible to buffer overflow" these same developers tend to call it a false positive. They're not trained to code to avoid problems, but rather to patch weaknesses. We need to encourage them to be effective software engineers and harden their code using CWE and CERT.

An interesting article called "Are vulnerabilities really a risk or simply hype?" explains that most application vulnerabilities aren't covered by firewalls, antivirus, etc. The ones they mention in the article are *all* quality issues, so security-only scanners don't necessarily look for them until the exploit is published! This is the "flavor-of-the-month" problem with most security tools. They are built like antivirus and look for known exploits rather than pointing out weaknesses in the code that should be hardened. The tester looks for flaws – the engineer looks to improve the code.

RISK AND PRIORITIZATION

One interesting and often-overlooked piece of CWE and CERT is that both standards have data for understanding risk and prioritizing the security defects they find. In the case of CWE this is called CWSS and Technical Impact. Basically, they help you understand that the underlying code problems detailed in CWE (like buffer overflow) lead to specific problems when exploited, like DoS or unexpected reading of protected data. Knowing what problems can happen helps you better decide which problems are most important in the context of your application or device. For CERT, there are scores for each rule and a recommendation that tells you how likely the problem is to occur in the real world, how difficult it is to mitigate, and the severity if it does happen. Together they get you a priority level that helps you focus on the most important issues first.

BEST APPROACH

So what's the first step? How can you get started? Now that you've decided to get ahead of the security problem by using static code analysis, you need to pick a tool. The tool should:

- Support all the security standards you need
- Support more than just the CWE Top 25

- Work with your code editors and IDEs
- Work with your build and CI tools
- Include flexible and comprehensive reporting
- Include both detection (flow/taint) rules as well as prevention rules (standards/pattern)
- Take advantage of the risk scoring algorithms in CWE and CERT to help you prioritize your security defects
- Can selectively execute based on current code, new code, changed code, and legacy code

Note that researchers at NIST and elsewhere have found that current open-source static analysis tools aren't as comprehensive as the available commercial tools, so make sure that the tool you use has you covered for what's important for your organization. OSS tools tend to be very narrow, finding just a few specific issues and lacking the ability to configure against real legacy code mixed with new, changing code.

Once you've selected a tool, you need to configure it to run the rules you want. Resist the urge to simply turn on all the security rules – it will destroy your long-term viability of static analysis. You're going to have more security findings than you can properly manage. Limit the number of rules in the beginning, and increase it as you bring the code into compliance.

An easy starting set is the CWE Top 25 – it will limit you to the most important issues. If you're using CERT, just use the rules that are set to the highest priority levels. Get developers used to addressing the issues and make sure they understand the relationship between rules that help prevent defects with rules that detect defects. As the code gets cleaner, add more rules.

In addition, it's worthwhile to do some root cause analysis on problems you're discovering during your security testing. If for example you're having SQLi issues, then look for all the rules related to SQLi and turn them on. If possible, craft a strategy that prevents SQLi defects in the first place and shore up all the weaknesses in your code.

A MODEST PROPOSAL

At Parasoft, we provide static analysis tools that support all of the major security coding standards like CWE and CERT, in addition to other industry standards like MISRA, JSF, & UL2900. On top of that, we've built a rich data-driven reporting system called Parasoft DTP, that helps you easily identify the most important issues out of the pool of possible problems you have. This system allows you to input from any Parasoft tools automatically as well as a host of other tools (both commercial and open source). It also has open REST APIs for both input and output, so you can easily integrate it into your build and developments systems, as well as software accounting systems of record for auditing.

It's important to remember that in a broad area like cybersecurity, you're not going to find everything you need from a single vendor, so having a way to integrate all your tools into a single process with a comprehensive report that takes into account all of your security activities is important for efficiency as well as compliance needs.

SUMMARY

As cybersecurity incidents increase, we can be sure that consumers will increasingly care more about the security of software and the devices they buy. At some point, it will not be surprising to see governments getting involved with various forms of regulation, such as requiring compliance with CWE and/or CERT. No matter where your software runs, whether on a desktop, mobile device, IoT device, or factory automation, software security is important and achievable with a rigorous standards-based development process.

www.parasoft.com

Parasoft Headquarters:
+1-626-256-3680

Parasoft EMEA:
+31-70-3922000

Parasoft APAC:
+65-6338-3628

PARASOFT®
Automated Software Testing