How Can I Enforce the SEI CERT C Coding Standard Using Static Analysis

Webinar
Agenda

• Why secure coding is a problem
• What is CERT
  • Why use it
  • how to use it
• C as a language is back!
• Embedded challenges for safety & security
• Secure-by-Design
• Tips, Tricks, and Traps
Internet of *(Insecure)* Things
Security can be tricky
Example: Cars are being hacked... because they talk too much

Most Vulnerabilities are Caused by Programming Errors

64% of the vulnerabilities in the NIST National Vulnerability Database due to programming errors
- 51% of those were due to classic errors like buffer overflows, cross-site scripting, injection flaws

Top vulnerabilities include
- Integer overflow
- Buffer overflow
- Missing authentication
- Missing or incorrect authorization
- Reliance on untrusted inputs (aka tainted inputs)

Sources: Heffley/Meunier: Can Source Code Auditing Software Identify Common Vulnerabilities and Be Used to Evaluate Software Security?
cwe.mitre.org/top25 Jan 6, 2015
The CERT C Coding Standard

Developed with community involvement since Spring 2008
- 1,568 registered experts on the wiki as of February 2014

Version 1.0 (C99) published by Addison-Wesley in September 2008

Version 2.0 was published in April 2014; extended for
- C11
- ISO/IEC TS 17961 Compatibility

Free PDF download published in 2016:
http://cert.org/secure-coding/products-services/secure-coding-download.cfm

“Current” guidelines available on CERT Secure Coding wiki
- https://www.securecoding.cert.org
CERT Rule Components 1

DCL22-CPP. Functions declared with [[noreturn]] must return void

Created by Aaron Ballman, last modified on Aug 24, 2016

As described in MSC55-CPP. Do not return from a function declared [[noreturn]]
attribute must not return on any code path. If a function declared with the
[[noreturn]] attribute has a non-void return value, it
implies that the function returns a value to the caller even though it would result in undefined behavior. Therefore, functions declared
with [[noreturn]] must also be declared as returning void.

Noncompliant Code Example

In this noncompliant code example, the function declared
[[noreturn]] int f() {
    std::exit(0);
    return 0;
}

This example does not violate MSC55-CPP. Do not return from a function declared [[noreturn]] because std::exit() is
declared [[noreturn]], so the return 0; statement can never be executed.
CERT Rule Components 2

Noncompliant Code Example
In this noncompliant code example, the function declared with \texttt{[[noreturn]]} claims to return an int:

```c
#include <stdlib>

[[noreturn]] int f() {
  std::exit(0);
  return 0;
}
```

This example does not violate MSC55-CPP: Do not return from a function declared \texttt{[[noreturn]]} because \texttt{std::exit()} is declared \texttt{[[noreturn]]}, so the return 0; statement can never be executed.

Compliant Solution
Because the function is declared \texttt{[[noreturn]]}, and no code paths in the function allow for a return in order to comply with MSC55-CPP: Do not return from a function declared \texttt{[[noreturn]]}, the compliant solution declares the function as returning void and elides the explicit return statement:

```c
#include <stdlib>

[[noreturn]] void f() {
  std::exit(0);
}
```

Noncompliant Code
\textit{Don’t try this at home!}

Compliant Code
\textit{Fixes noncompliant code.}
CERT Rule Components

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clang</td>
<td>3.9</td>
<td>-Winvalid-noreturn</td>
<td></td>
</tr>
</tbody>
</table>

Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the CERT website.

Related Guidelines

| SEI CERT C++ Coding Standard | MSC54-CPP. Value-returning functions must return a value from all exit paths MSC55-CPP. Do not return from a function declared [[noretturn]] |

Bibliography

CERT Rule Components 4

Risk Assessment

A function declared with a non-void return type and declared with the \[[noreturn]\] attribute is confusing to consumers of the function because the two declarations are conflicting. In turn, it can result in misuse of the API by the consumer or can indicate an implementation bug by the producer.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>DCL22-CPP</td>
<td>Low</td>
<td>Unlikely</td>
<td>Low</td>
<td>P3</td>
<td>L3</td>
</tr>
</tbody>
</table>

Automated Detection

<table>
<thead>
<tr>
<th>Tool</th>
<th>Version</th>
<th>Checker</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clang</td>
<td>3.9</td>
<td>-Winvalid-noreturn</td>
<td></td>
</tr>
</tbody>
</table>
Risk Assessment

Risk assessment is performed using failure mode, effects, and criticality analysis.

<table>
<thead>
<tr>
<th>Severity</th>
<th>Value</th>
<th>Meaning</th>
<th>Examples of Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>low</td>
<td>1</td>
<td>low</td>
<td>denial-of-service attack, abnormal termination</td>
</tr>
<tr>
<td>medium</td>
<td>2</td>
<td>medium</td>
<td>data integrity violation, unintentional information disclosure</td>
</tr>
<tr>
<td>high</td>
<td>3</td>
<td>high</td>
<td>run arbitrary code</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Likelihood</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>unlikely</td>
<td>1</td>
<td>unlikely</td>
</tr>
<tr>
<td>probable</td>
<td>2</td>
<td>probable</td>
</tr>
<tr>
<td>likely</td>
<td>3</td>
<td>likely</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cost</th>
<th>Value</th>
<th>Meaning</th>
<th>Detection</th>
<th>Correction</th>
</tr>
</thead>
<tbody>
<tr>
<td>high</td>
<td>1</td>
<td>high</td>
<td>manual</td>
<td>manual</td>
</tr>
<tr>
<td>medium</td>
<td>2</td>
<td>medium</td>
<td>automatic</td>
<td>manual</td>
</tr>
<tr>
<td>low</td>
<td>3</td>
<td>low</td>
<td>automatic</td>
<td>automatic</td>
</tr>
</tbody>
</table>
Levels and Priorities

- **L1: P12 – P27**
  - High severity, likely, inexpensive to repair flaws

- **L2: P6 – P9**
  - Medium severity, probable, medium cost to repair flaws

- **L3: P1 – P4**
  - Low severity, unlikely, expensive to repair flaws
Degrees of Severity

CIA Triad:
- Confidentiality
- Integrity
- Availability

CERT Severity Levels:

<table>
<thead>
<tr>
<th>Severity</th>
<th>Meaning</th>
<th>Examples of Vulnerability</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>low</td>
<td>denial-of-service attack, abnormal termination</td>
</tr>
<tr>
<td>2</td>
<td>medium</td>
<td>unintentional information disclosure</td>
</tr>
<tr>
<td>3</td>
<td>high</td>
<td>run arbitrary code, privilege escalation</td>
</tr>
</tbody>
</table>

**Severity**—How serious are the consequences of the rule being ignored?
### 2011 CWE/SANS Top 25 Most Dangerous Software Errors

<table>
<thead>
<tr>
<th>Rank</th>
<th>Score</th>
<th>ID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>[1]</td>
<td>93.8</td>
<td>CWE-89</td>
<td>Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')</td>
</tr>
<tr>
<td>[2]</td>
<td>83.3</td>
<td>CWE-78</td>
<td>Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')</td>
</tr>
<tr>
<td>[3]</td>
<td>79.0</td>
<td>CWE-120</td>
<td>Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')</td>
</tr>
<tr>
<td>[4]</td>
<td>77.7</td>
<td>CWE-79</td>
<td>Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')</td>
</tr>
<tr>
<td>[6]</td>
<td>76.8</td>
<td>CWE-862</td>
<td>Missing Authorization</td>
</tr>
<tr>
<td>[7]</td>
<td>75.0</td>
<td>CWE-798</td>
<td>Use of Hard-coded Credentials</td>
</tr>
<tr>
<td>[8]</td>
<td>75.0</td>
<td>CWE-311</td>
<td>Missing Encryption of Sensitive Data</td>
</tr>
<tr>
<td>[9]</td>
<td>74.0</td>
<td>CWE-434</td>
<td>Unrestricted Upload of File with Dangerous Type</td>
</tr>
<tr>
<td>[10]</td>
<td>73.8</td>
<td>CWE-807</td>
<td>Reliance on Untrusted Inputs in a Security Decision</td>
</tr>
</tbody>
</table>

[http://cwe.mitre.org/top25/#Listing](http://cwe.mitre.org/top25/#Listing)
C is the primary language of embedded

Fastest growing language 2017 (Tiobe)
Top in employer demand and growth (IEEE Spectrum)
Also C++

![Graph showing programming language use in embedded systems](image-url)
Barr Group Embedded Security Safety Report 2017 & 2018

Secure Coding Practices Adoption %

- Coding Standards
- Code Review
- Static Analysis
- Standards no Static

- 2017
- 2018
- not enforced 18
POLL: Secure Coding Standards

What coding standards do you use?

CERT
CWE
MISRA
OTHER
NONE
Barr Group Survey 2018:
Coding standards used for embedded safety-critical

Figure 15. Primary Bases for Coding Standards Used in Safety-Critical Products

Safety standards
Not security
Fix or Prevent

Secure-by-design is a movement to create software that is secure rather than trying to test security into software. By-design is a requirement of GDPR for privacy and security.

“Although the notion of protecting software is an important one, it’s just plain **easier to protect something that is defect-free** than something riddled with vulnerabilities.”

(Gary McGraw, Cigital)
Policy first

- What teams need to do SA?
- What projects require SA?
- What rules are required?
- What amount of compliance?
- When can you suppress?
- How to handle legacy code?
- Do you ship with SA violations?
  - Rules / recommendations?
  - Levels?
Training

• Secure coding basics
• Hacking
• How to use & interpret standards
• IMPORTANCE of security
Workflow Demo

Support for IDE

Support for servers and CI/CD with enforced same configuration
Noise and perceptions

- “Static analysis is a pain”
- False positives has varying definitions
  - I don’t like it
  - It was wrong
- True false positives in pattern rules means rule deficiency
- Context
  - Does this apply here and now?
  - In-code suppressions to document decision
- Flow analysis style False positives are inevitable
  - Finds real bugs
  - Flow analysis is not comprehensive
Getting the configuration right

• Rules vs Recommendations
• Severity & Priority levels
• Static Analysis is about process, It’s incremental
• Avoid biting off more than you can chew

High severity, likely, inexpensive to repair flaws
Medium severity, probable, medium cost to repair flaws
Low severity, unlikely, expensive to repair flaws
## Select SCALe Assessments

<table>
<thead>
<tr>
<th>Codebase</th>
<th>Date</th>
<th>Customer</th>
<th>Lang</th>
<th>ksLOC</th>
<th>Rules</th>
<th>Diags</th>
<th>True</th>
<th>Suspect</th>
<th>Diag /KsLOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>6/12</td>
<td>Gov1</td>
<td>C++</td>
<td>38.8</td>
<td>12</td>
<td>1,07 1</td>
<td>52</td>
<td>1,019</td>
<td>27.6</td>
</tr>
<tr>
<td>B</td>
<td>3/13</td>
<td>Gov1</td>
<td>C</td>
<td>87.4</td>
<td>28</td>
<td>17,5 43</td>
<td>86</td>
<td>17,457</td>
<td>200.7</td>
</tr>
<tr>
<td>C</td>
<td>10/13</td>
<td>Gov2</td>
<td>C</td>
<td>9,585</td>
<td>18</td>
<td>289</td>
<td>159</td>
<td>130</td>
<td>0.03</td>
</tr>
<tr>
<td>D</td>
<td>6/12</td>
<td>Gov3</td>
<td>Java</td>
<td>4.27</td>
<td>18</td>
<td>345</td>
<td>117</td>
<td>228</td>
<td>80.8</td>
</tr>
<tr>
<td>E</td>
<td>9/12</td>
<td>Gov2</td>
<td>Java</td>
<td>61.2</td>
<td>33</td>
<td>538</td>
<td>288</td>
<td>250</td>
<td>8.8</td>
</tr>
<tr>
<td>F</td>
<td>11/13</td>
<td>Gov2</td>
<td>Java</td>
<td>17.6</td>
<td>21</td>
<td>414</td>
<td>341</td>
<td>73</td>
<td>23.5</td>
</tr>
<tr>
<td>G</td>
<td>2/14</td>
<td>Gov4</td>
<td>Java</td>
<td>653</td>
<td>29</td>
<td>8,52 6</td>
<td>64</td>
<td>8,462</td>
<td>13.1</td>
</tr>
<tr>
<td>H</td>
<td>3/14</td>
<td>Gov5</td>
<td>Java</td>
<td>1.51</td>
<td>8</td>
<td>53</td>
<td>53</td>
<td>0</td>
<td>35.1</td>
</tr>
<tr>
<td>I</td>
<td>5/14</td>
<td>Mil1</td>
<td>Java</td>
<td>403</td>
<td>27</td>
<td>3114 723</td>
<td>2,391</td>
<td>7.7</td>
<td></td>
</tr>
<tr>
<td>J</td>
<td>1/11</td>
<td>Gov3</td>
<td>Perl</td>
<td>93.6</td>
<td>36</td>
<td>6,92 5</td>
<td>357</td>
<td>6,568</td>
<td>74.0</td>
</tr>
<tr>
<td>K</td>
<td>5/14</td>
<td>Gov3</td>
<td>Perl</td>
<td>10.2</td>
<td>10</td>
<td>133</td>
<td>84</td>
<td>49</td>
<td>13.0</td>
</tr>
</tbody>
</table>
Parasoft CERT C/C++ Solution DEMO

Complete support for CERT-C-RULES

CERT centric
- Rule names, dashboards and reports

CERT Risk score
- Likelihood, cost, priority
- a technical edge
Conclusions

• Security in IoT is extremely important, especially where safety is at stake

• Security is achievable if you take a proactive approach rather than trying to test security in

• Tools and process are both important to a successful SAST initiative