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

Webinar

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, PA 15213

This material is based upon work funded and supported by the Department of Defense under Contract No. FA8702-15-D-0002 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center.

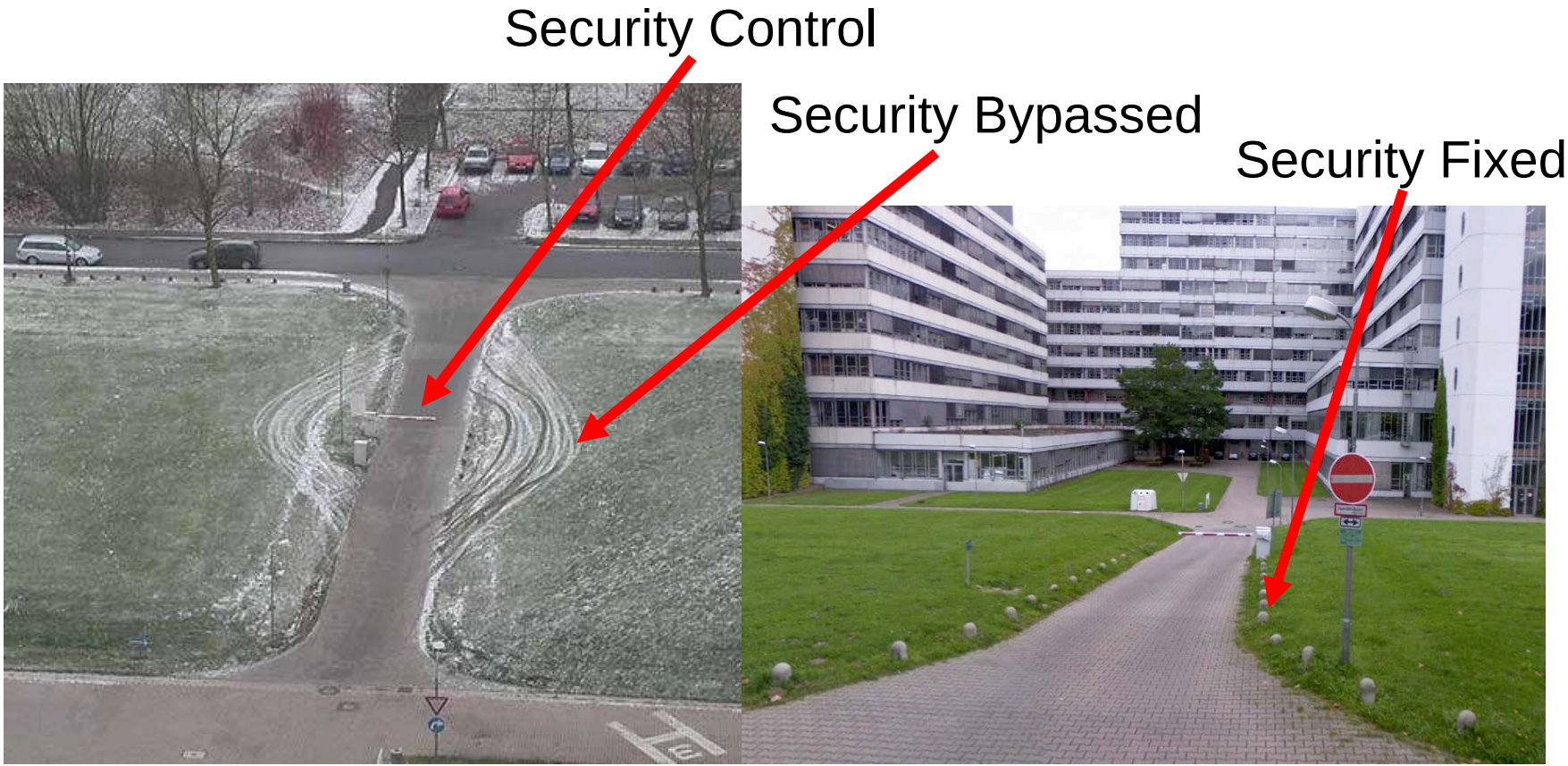
DM17-0546

Agenda

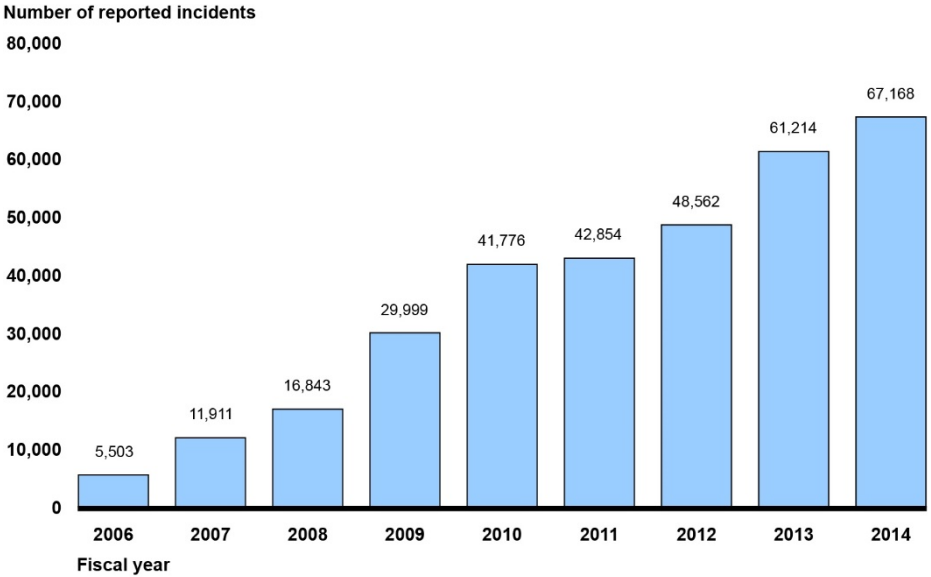
- Why secure coding is a problem
- What is CERT
  - Why use it
  - how to use it
- C as a language is back!
- Embedded challenges for safety & security
- Secure-by-Design
- Tips, Tricks, and Traps

Internet of *(Insecure)* Things

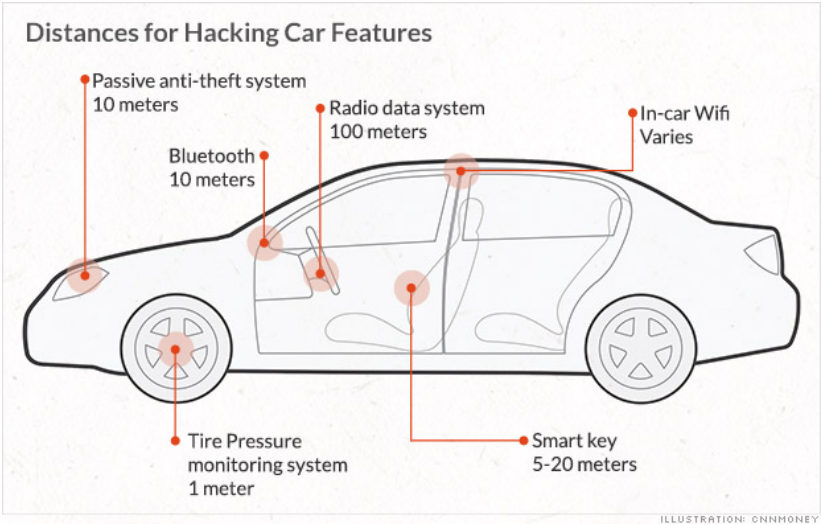
Security can be tricky



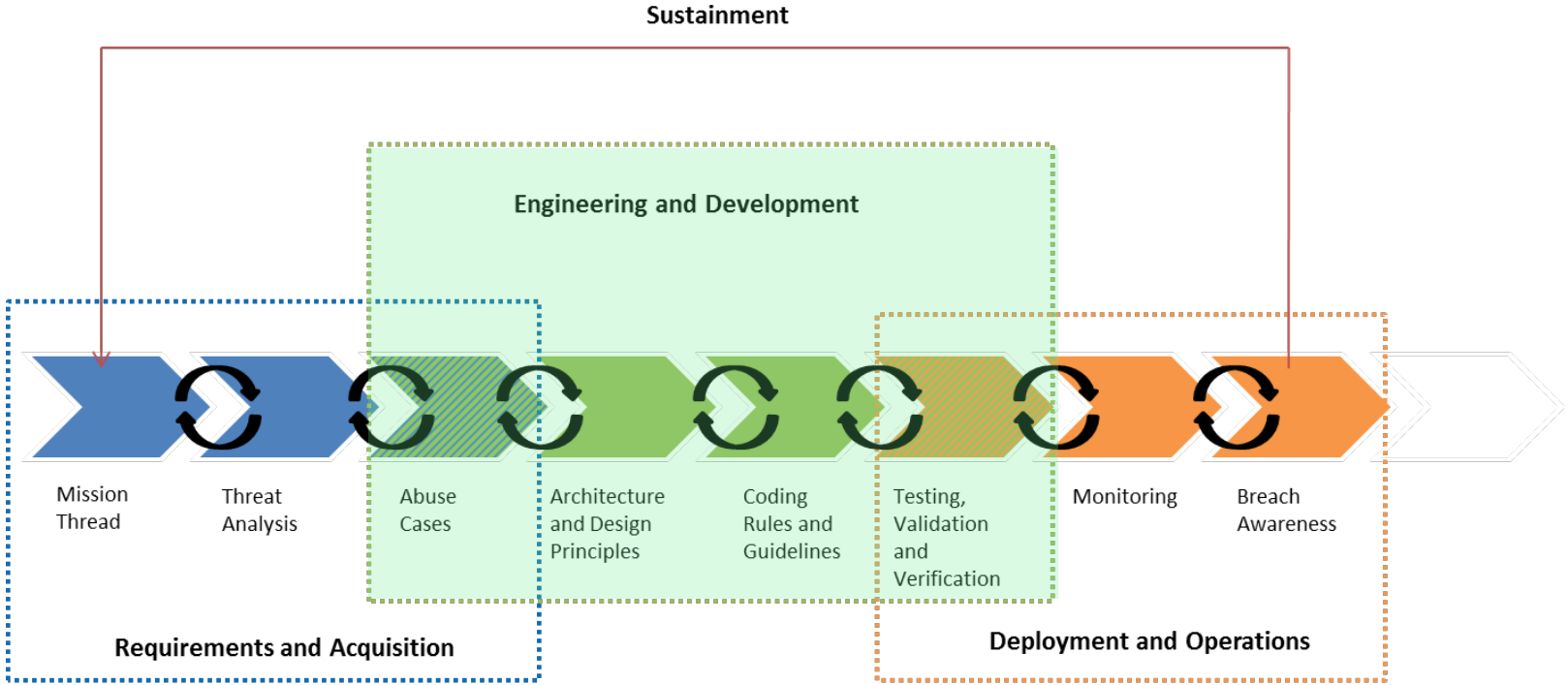
Example: Cars are being hacked… because they talk too much



Source: GAO analysis of United States Computer Emergency Readiness Team data for fiscal years 2006-2014. | GAO-15-573T



Engineering and Development



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

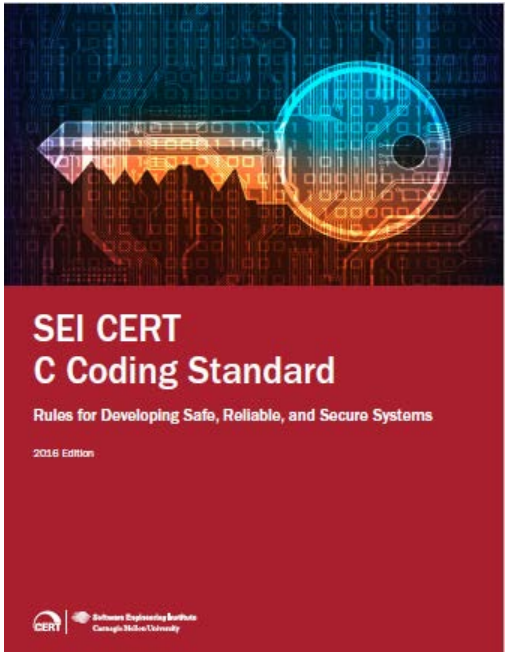
- C11
- ISO/IEC TS 17961 Compatibility

Free PDF download published in 2016:

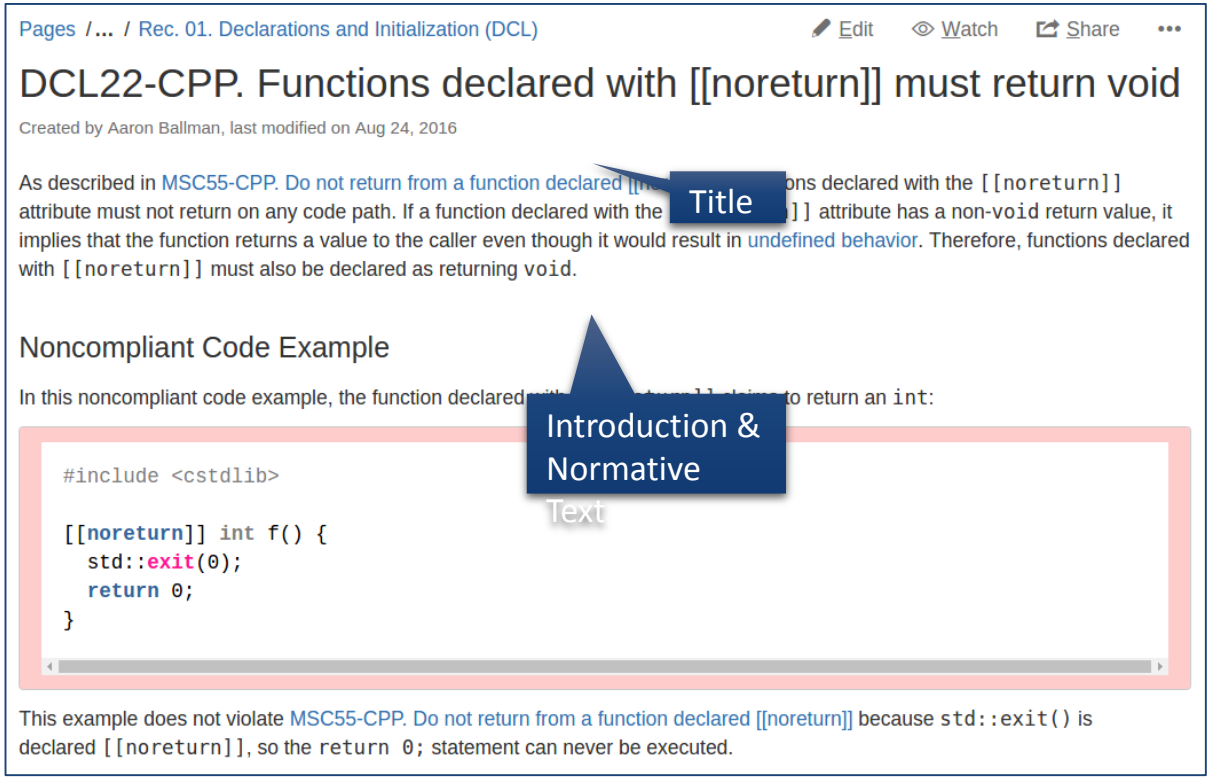
http://cert.org/secure-coding/products-services/secure-coding-download.cfm

“Current” guidelines available on CERT Secure Coding wiki

- https://www.securecoding.cert.org



CERT Rule Components 1



CERT Rule Components 2

Noncompliant Code Example

In this noncompliant code example, the function declared with `[[noreturn]]` claims to return an `int`:

```
#include <cstdlib>

[[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.

Compliant Solution

Because the function is declared `[[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 [[noreturn]], the compliant solution declares the function as returning `void` and elides the explicit `return` statement:

```
#include <cstdlib>

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

Noncompliant Code
Don't try this at home!

Compliant Code
Fixes noncompliant code.

CERT Rule Components 3

Automated Detection

Tool	Version	Checker	Description
Clang	3.9	`-Winvalid-noreturn`	

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 [[noreturn]]

Bibliography

[ISO/IEC 14882-2014]	Subclause 7.6.3, "Noreturn Attribute"

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.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
DCL22-CPP	Low	Unlikely	Low	P3	L3

Automated Detection

Tool	Version	Checker	Description
Clang	3.9	`-Winvalid-noreturn`	

Risk Assessment

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

Severity—How serious are the consequences of the rule being ignored?

Value	Meaning	Examples of Vulnerability
1	low	denial-of-service attack, abnormal termination
2	medium	data integrity violation, unintentional information disclosure
3	high	run arbitrary code

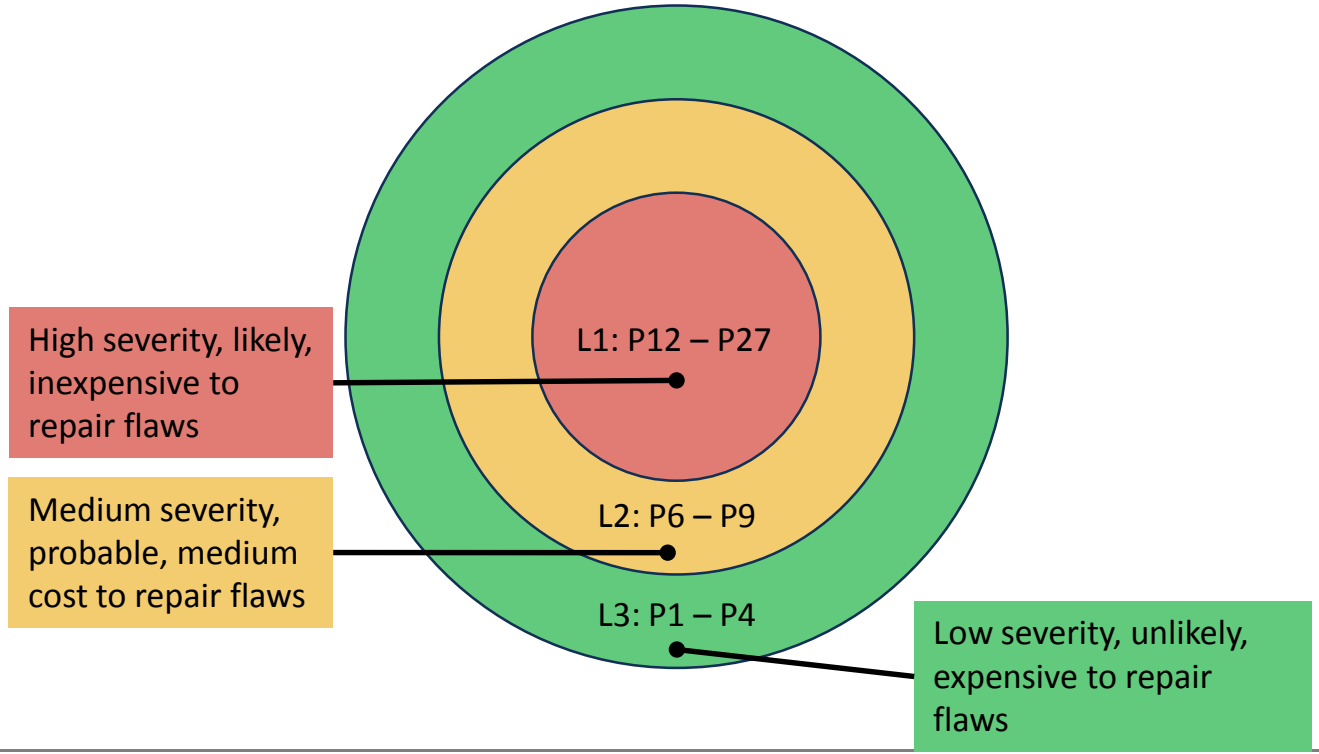
Likelihood—How likely is it that a flaw introduced by ignoring the rule can lead to an exploitable vulnerability?

Value	Meaning
1	unlikely
2	probable
3	likely

Cost—The cost of mitigating the vulnerability.

Value	Meaning	Detection	Correction
1	high	manual	manual
2	medium	automatic	manual
3	low	automatic	automatic

Levels and Priorities



Degrees of Severity

CIA Triad:

- Confidentiality
- Integrity
- Availability

CERT Severity Levels:

Severity—How serious are the consequences of the rule being ignored?

Value	Meaning	Examples of Vulnerability
1	low	denial-of-service attack, abnormal termination
2	medium	unintentional information disclosure
3	high	run arbitrary code, privilege escalation

2011 CWE/SANS Top 25 Most Dangerous Software Errors

Rank	Score	ID	Name
[1]	93.8	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')
[2]	83.3	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
[3]	79.0	CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')
[4]	77.7	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')
[5]	76.9	CWE-306	Missing Authentication for Critical Function
[6]	76.8	CWE-862	Missing Authorization
[7]	75.0	CWE-798	Use of Hard-coded Credentials
[8]	75.0	CWE-311	Missing Encryption of Sensitive Data
[9]	74.0	CWE-434	Unrestricted Upload of File with Dangerous Type
[10]	73.8	CWE-807	Reliance on Untrusted Inputs in a Security Decision

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++

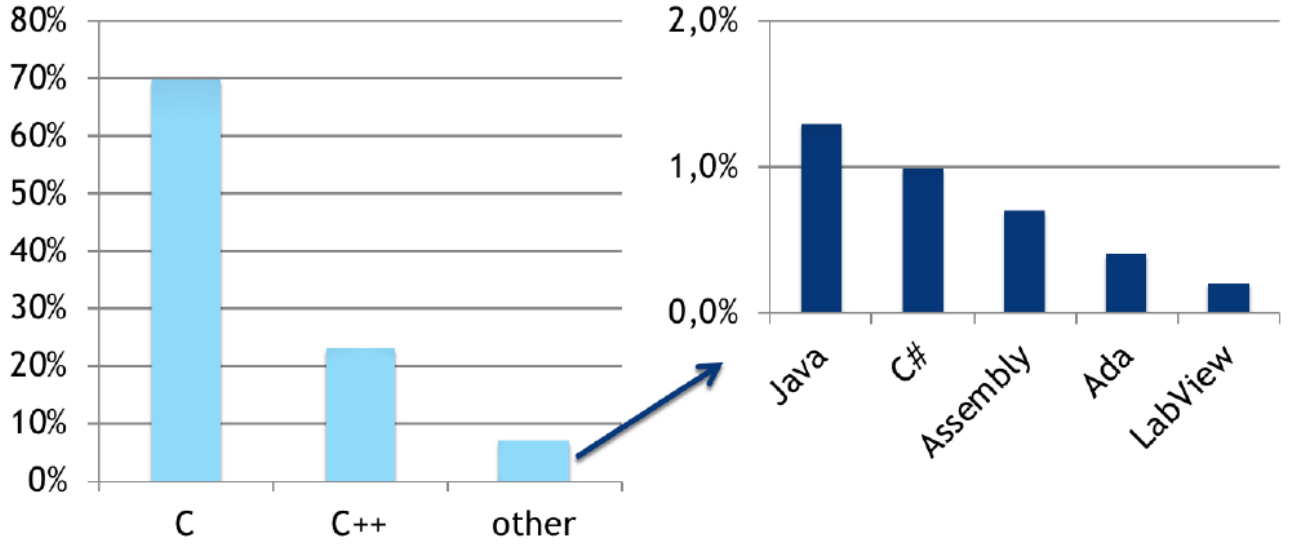
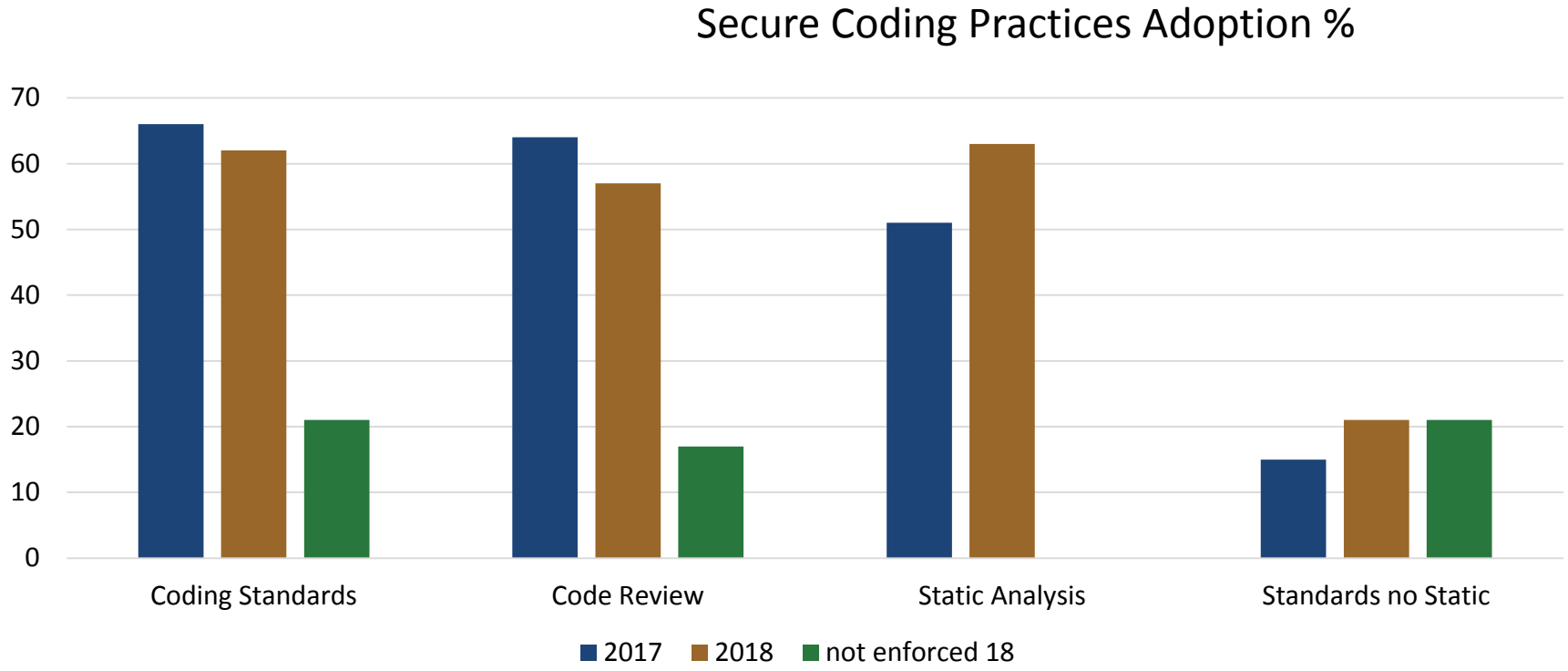


Figure 11. Primary Programming Language in Embedded Systems Designs

Barr Group Embedded Survey 2018

Barr Group Embedded Security Safety Report 2017 & 2018



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

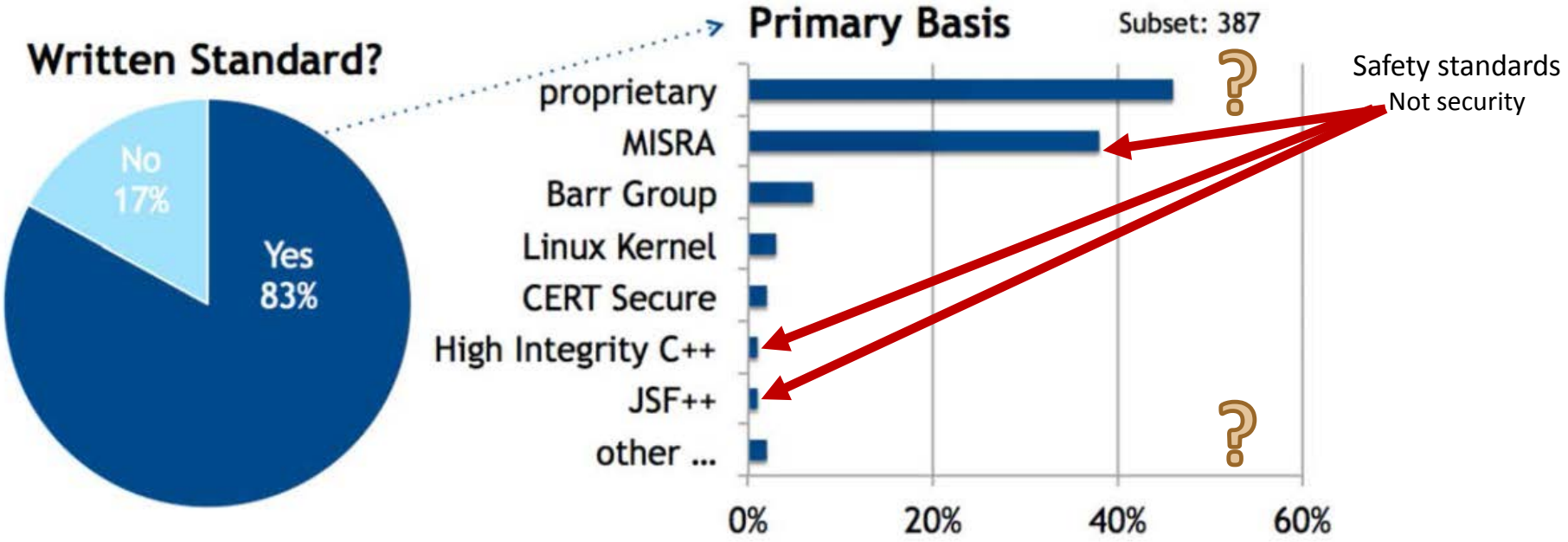


Figure 15. Primary Bases for Coding Standards Used in Safety-Critical Products

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?

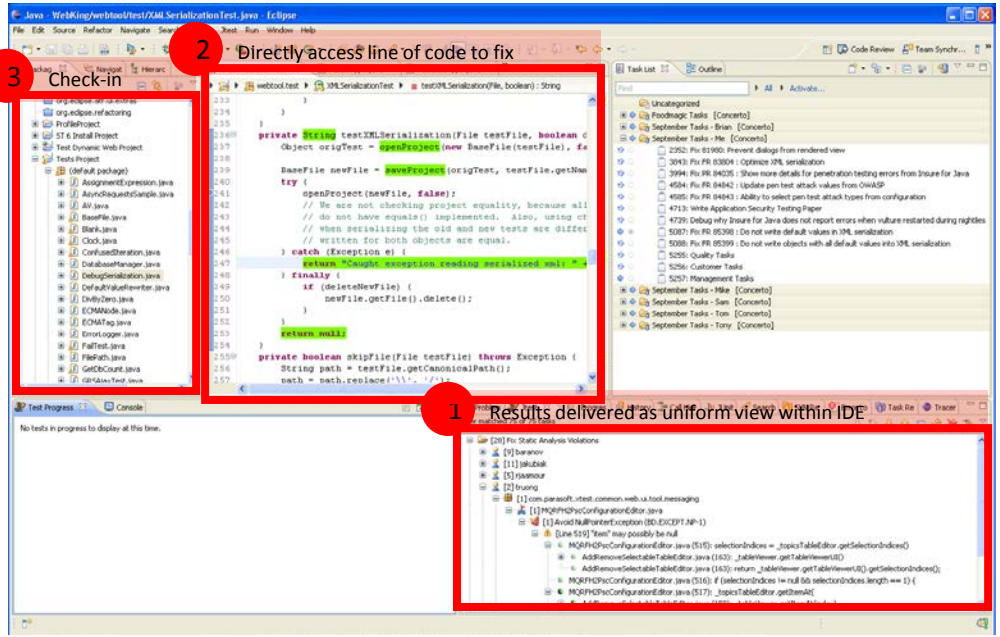
Training

- Secure coding basics
- Hacking
- How to use & interpret standards
- IMPORTANCE of security

Workflow Demo

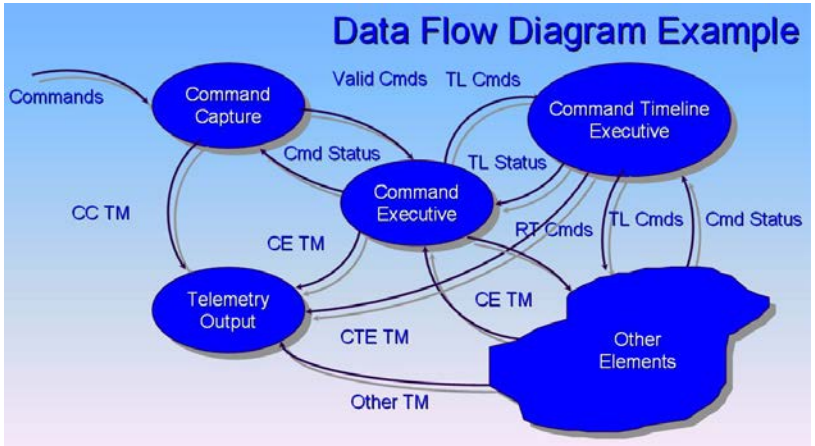
Support for IDE

Support for servers and CI/CD with enforced same configuration



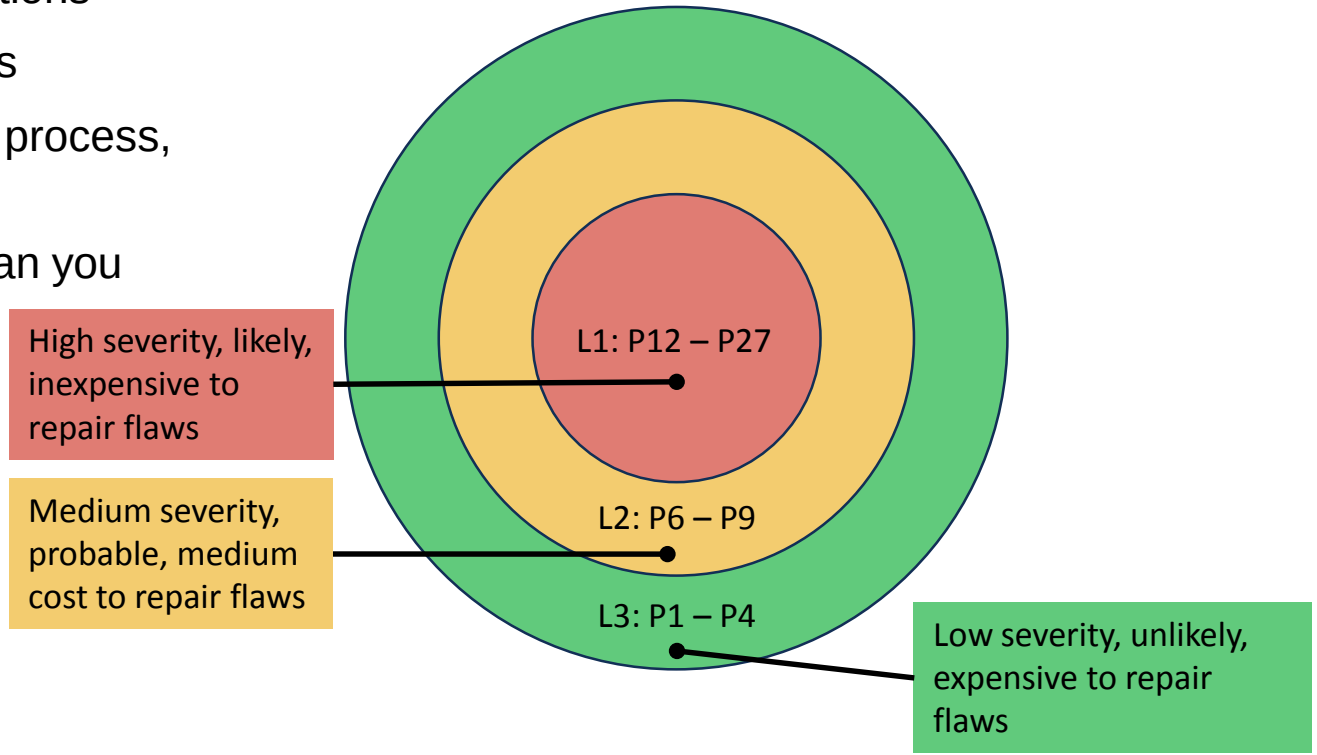
Noise and perceptions

- “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



Select SCALe Assessments

Codebase	Date	Customer	Lang	ksLOC	Rules	Diags	True	Suspect	Diag /KsLOC
A	6/12	Gov1	C++	38.8	12	1,071	52	1,019	27.6
B	3/13	Gov1	C	87.4	28	17,543	86	17,457	200.7
C	10/13	Gov2	C	9,585	18	289	159	130	0.03
D	6/12	Gov3	Java	4.27	18	345	117	228	80.8
E	9/12	Gov2	Java	61.2	33	538	288	250	8.8
F	11/13	Gov2	Java	17.6	21	414	341	73	23.5
G	2/14	Gov4	Java	653	29	8,526	64	8,462	13.1
H	3/14	Gov5	Java	1.51	8	53	53	0	35.1
I	5/14	Mil1	Java	403	27	3114	723	2,391	7.7
J	1/11	Gov3	Perl	93.6	36	6,925	357	6,568	74.0
K	5/14	Gov3	Perl	10.2	10	133	84	49	13.0

Parasoft CERT C/C++ Solution DEMO

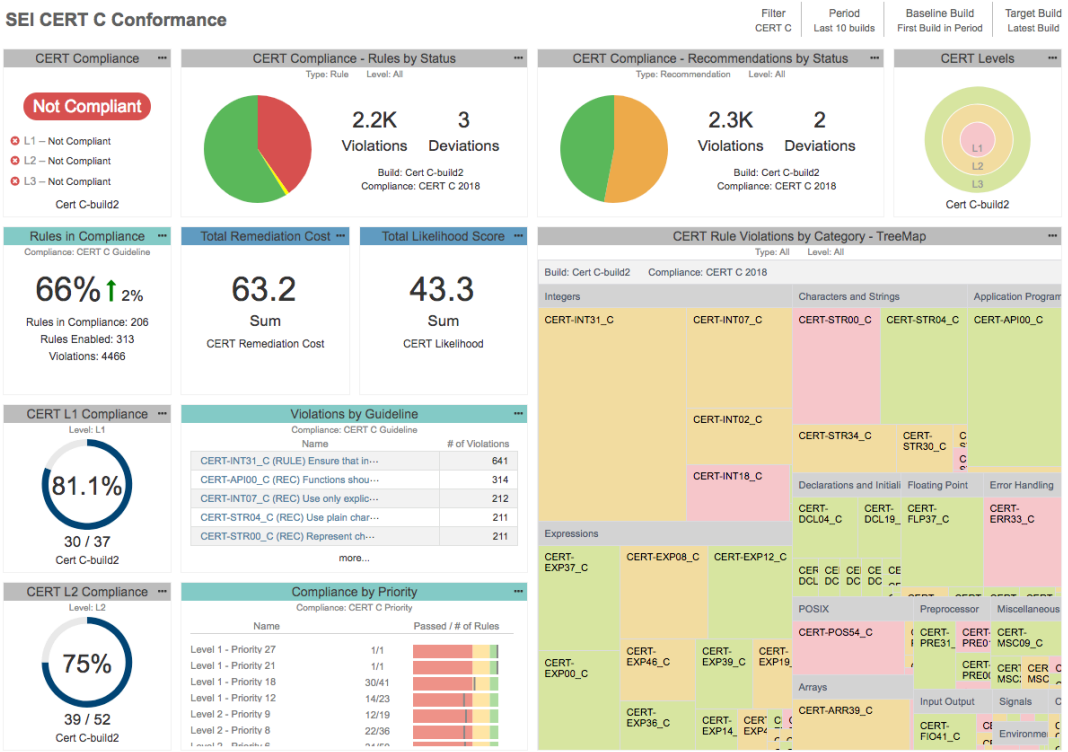
Complete support for CERT-C-RULES

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

For More Information

David Svoboda
Software Security Engineer
Secure Coding Initiative
Phone: (412) 268-3965
Email: svoboda@cert.org

Arthur Hicken
Evangelist
Parasoft
Phone: (626) 275-2445
Email: codecurmudgeon@parasoft.com

Web
www.cert.org/secure-coding
www.securecoding.cert.org (wiki)

Web
www.parasoft.com