MATLAB EXPO 2021

자동차 사이버보안: UN-ECE WP.29 및 ISO 21434에서 정적 코드 분석의 역할

유용출

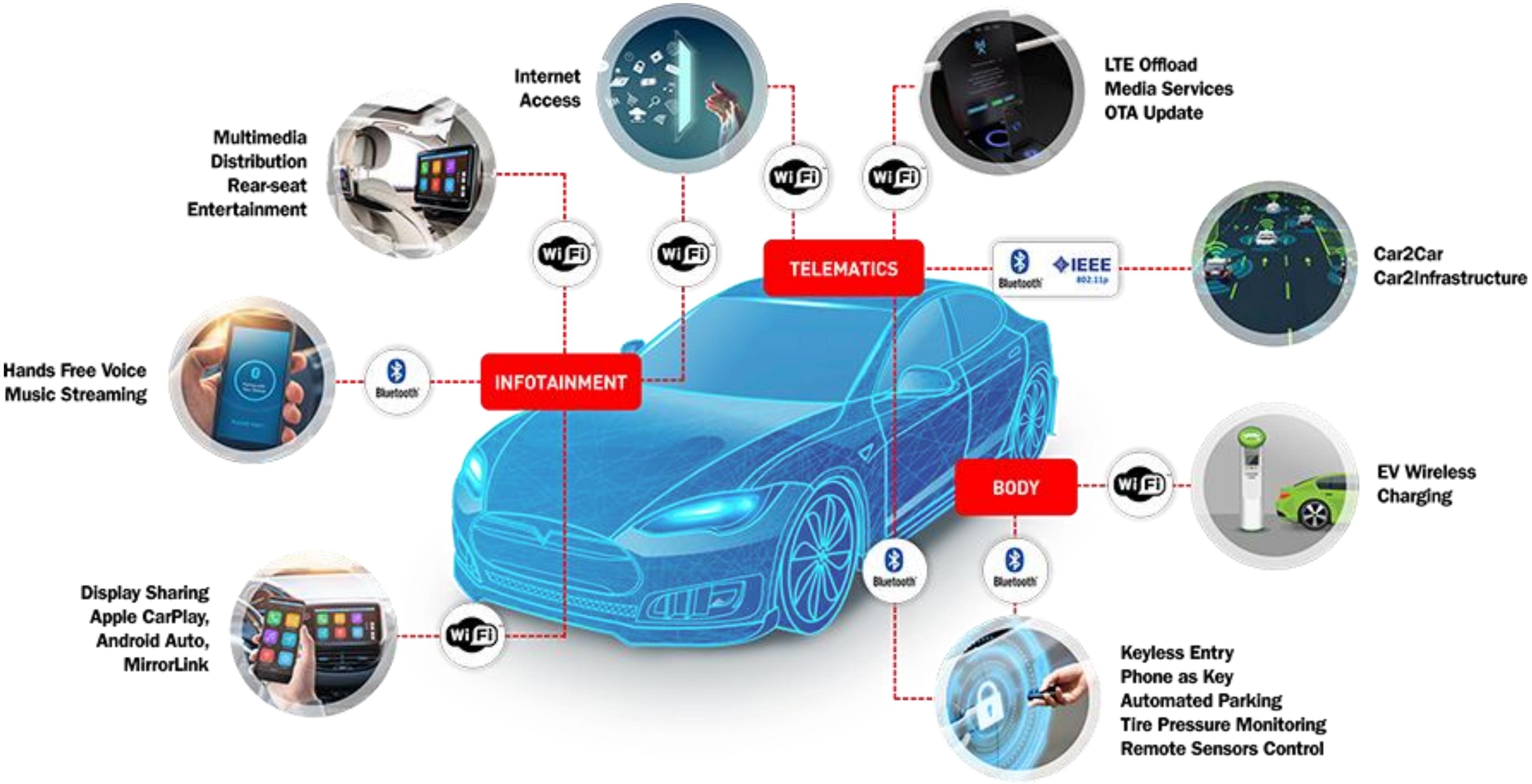
MathWorks®

Agenda

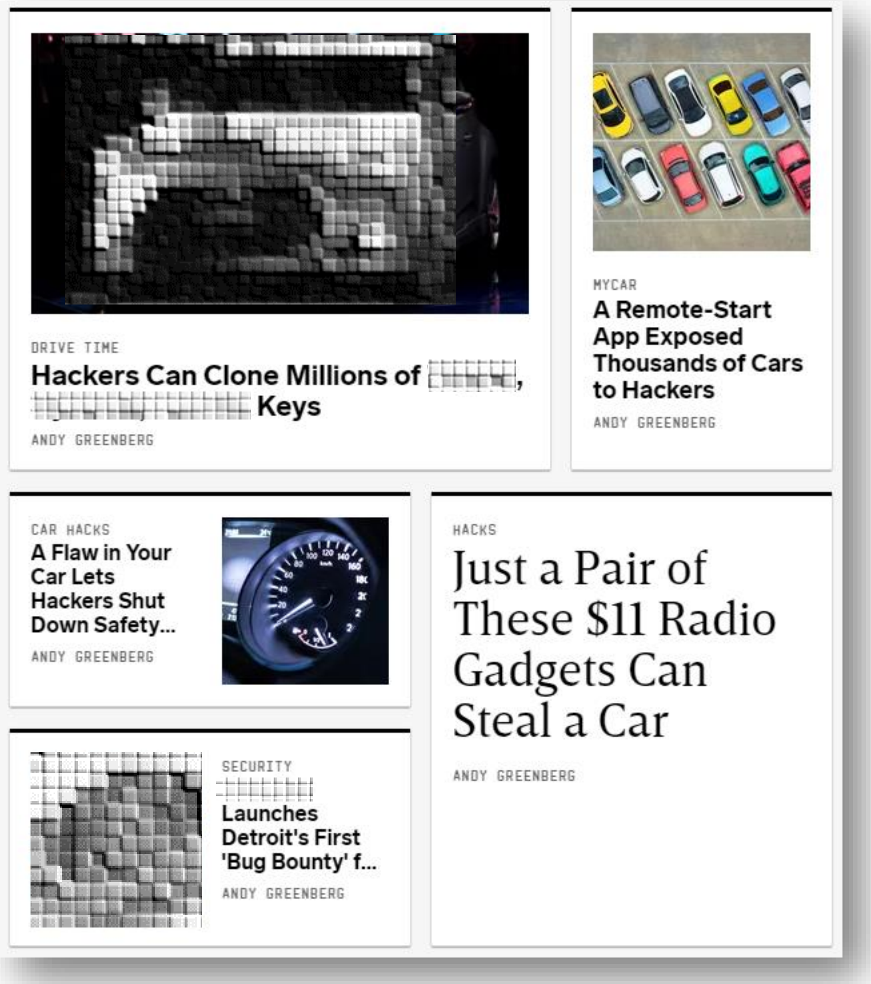
- Cybersecurity - News, Regulations and Standards
- Automotive Cybersecurity & Static Application Security Testing
- Catching Up with Cybersecurity in Three Steps

Cybersecurity –
News, Regulations and Standards

Vehicle Connectivity



Automotive Cybersecurity in the News



DRIVE TIME

Hackers Can Clone Millions of [redacted], [redacted] Keys

ANDY GREENBERG

MYCAR

A Remote-Start App Exposed Thousands of Cars to Hackers

ANDY GREENBERG

CAR HACKS

A Flaw in Your Car Lets Hackers Shut Down Safety...

ANDY GREENBERG

HACKS

Just a Pair of These $11 Radio Gadgets Can Steal a Car

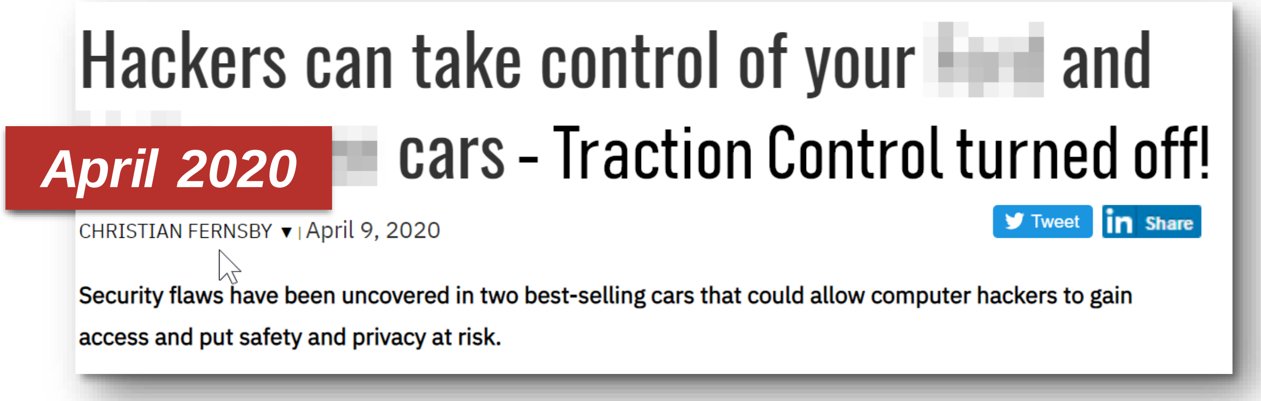
ANDY GREENBERG

SECURITY

[redacted] Launches Detroit's First 'Bug Bounty' f...

ANDY GREENBERG

https://www.wired.com/tag/car-hacking/



Hackers can take control of your [redacted] and [redacted] cars - Traction Control turned off!

April 2020

CHRISTIAN FERNSBY ▾ | April 9, 2020

Tweet in Share

Security flaws have been uncovered in two best-selling cars that could allow computer hackers to gain access and put safety and privacy at risk.

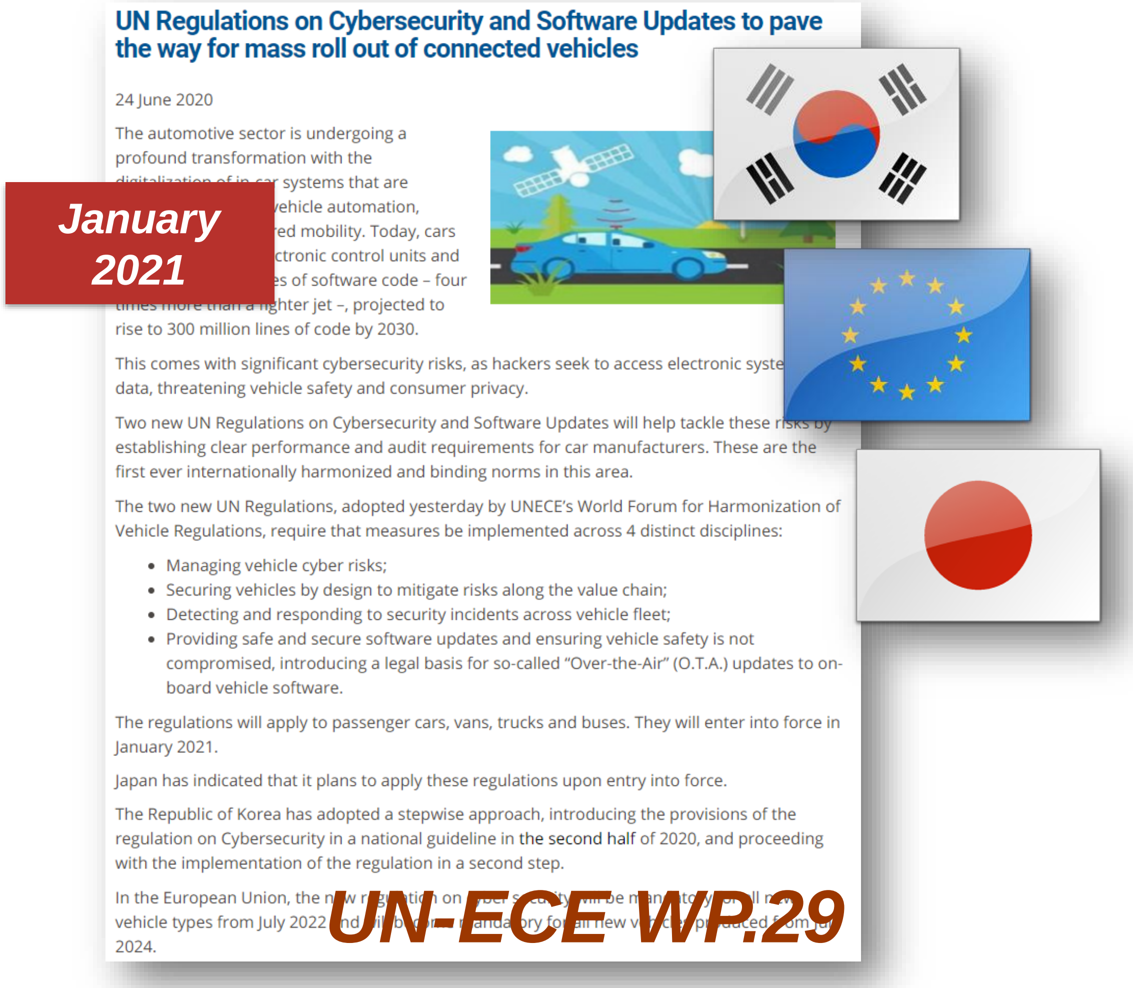
https://www.poandpo.com/news/hackers-can-take-control-of-your-ford-and-volkswagen-cars-942020422/

Vehicle remote control

Privacy breach

Vehicle theft

New Regulations and Guidance



UN Regulations on Cybersecurity and Software Updates to pave the way for mass roll out of connected vehicles

24 June 2020

The automotive sector is undergoing a profound transformation with the digitalization of in-car systems that are ... vehicle automation, ... ed mobility. Today, cars ... ctronic control units and ... es of software code – four times more than a fighter jet –, projected to rise to 300 million lines of code by 2030.

This comes with significant cybersecurity risks, as hackers seek to access electronic syste... data, threatening vehicle safety and consumer privacy.

Two new UN Regulations on Cybersecurity and Software Updates will help tackle these risks by establishing clear performance and audit requirements for car manufacturers. These are the first ever internationally harmonized and binding norms in this area.

The two new UN Regulations, adopted yesterday by UNECE's World Forum for Harmonization of Vehicle Regulations, require that measures be implemented across 4 distinct disciplines:

- Managing vehicle cyber risks;
- Securing vehicles by design to mitigate risks along the value chain;
- Detecting and responding to security incidents across vehicle fleet;
- Providing safe and secure software updates and ensuring vehicle safety is not compromised, introducing a legal basis for so-called "Over-the-Air" (O.T.A.) updates to on-board vehicle software.

The regulations will apply to passenger cars, vans, trucks and buses. They will enter into force in January 2021.

Japan has indicated that it plans to apply these regulations upon entry into force.

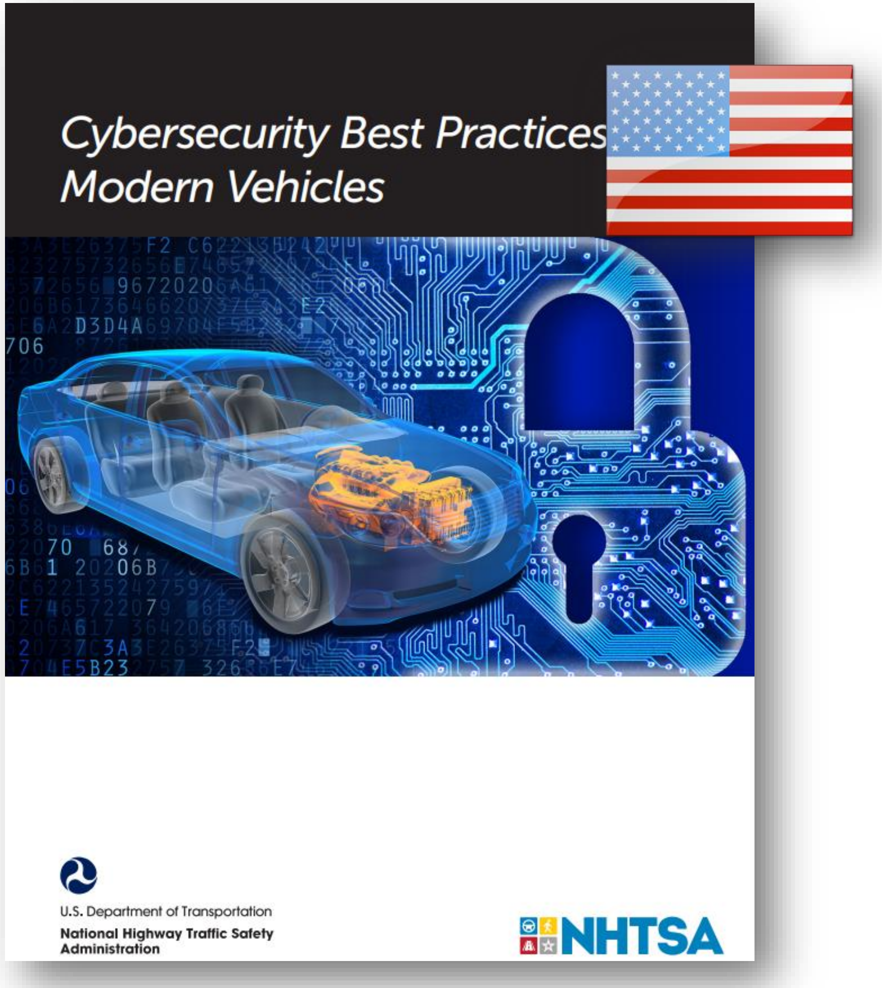
The Republic of Korea has adopted a stepwise approach, introducing the provisions of the regulation on Cybersecurity in a national guideline in the second half of 2020, and proceeding with the implementation of the regulation in a second step.

In the European Union, the new regulation on cyber security will be mandatory for all new vehicle types from July 2022 and will become mandatory for all new vehicles produced from July 2024.

January 2021

UN-ECE WP.29

https://unece.org/press/un-regulations-cybersecurity-and-software-updates-pave-way-mass-roll-out-connected-vehicles



Cybersecurity Best Practices for Modern Vehicles

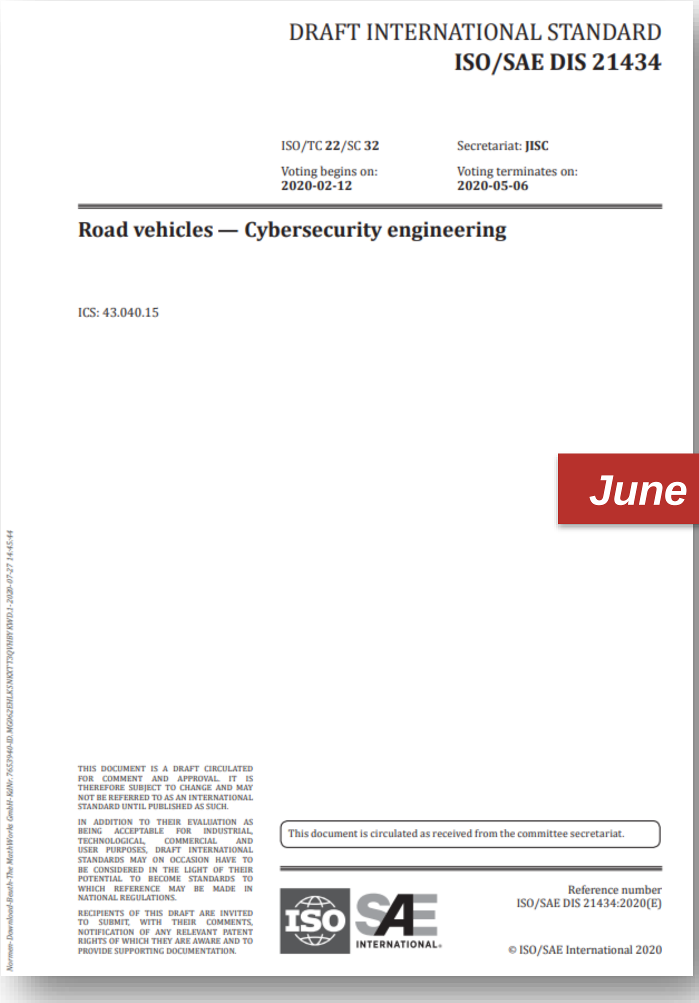
U.S. Department of Transportation
National Highway Traffic Safety Administration

NHTSA

https://www.nhtsa.gov/staticfiles/nvs/pdf/812333_CybersecurityForModernVehicles.pdf

New Standards

ISO/SAE 21434 - Road vehicles — Cybersecurity engineering



DRAFT INTERNATIONAL STANDARD
ISO/SAE DIS 21434

ISO/TC **22**/SC **32**
Secretariat: **JISC**

Voting begins on:
2020-02-12

Voting terminates on:
2020-05-06

Road vehicles — Cybersecurity engineering

ICS: 43.040.15

THIS DOCUMENT IS A DRAFT CIRCULATED FOR COMMENT AND APPROVAL. IT IS THEREFORE SUBJECT TO CHANGE AND MAY NOT BE REFERRED TO AS AN INTERNATIONAL STANDARD UNTIL PUBLISHED AS SUCH.

IN ADDITION TO THEIR EVALUATION AS BEING ACCEPTABLE FOR INDUSTRIAL, TECHNOLOGICAL, COMMERCIAL AND USER PURPOSES, DRAFT INTERNATIONAL STANDARDS MAY ON OCCASION HAVE TO BE CONSIDERED IN THE LIGHT OF THEIR POTENTIAL TO BECOME STANDARDS TO WHICH REFERENCE MAY BE MADE IN NATIONAL REGULATIONS.

RECIPIENTS OF THIS DRAFT ARE INVITED TO SUBMIT, WITH THEIR COMMENTS, NOTIFICATION OF ANY RELEVANT PATENT RIGHTS OF WHICH THEY ARE AWARE AND TO PROVIDE SUPPORTING DOCUMENTATION.

This document is circulated as received from the committee secretariat.

Reference number
ISO/SAE DIS 21434:2020(E)

June 2021

- Standard for Auto industry – ISO 26262 cybersecurity counterpart
- Can be used as reference standard WP.29 and NHTSA

UN Vehicle Regulations Enter into Force

UNITED NATIONS
ECONOMIC COMMISSION
FOR EUROPE

The following standards may be applicable:

(a) **ISO/SAE 21434**

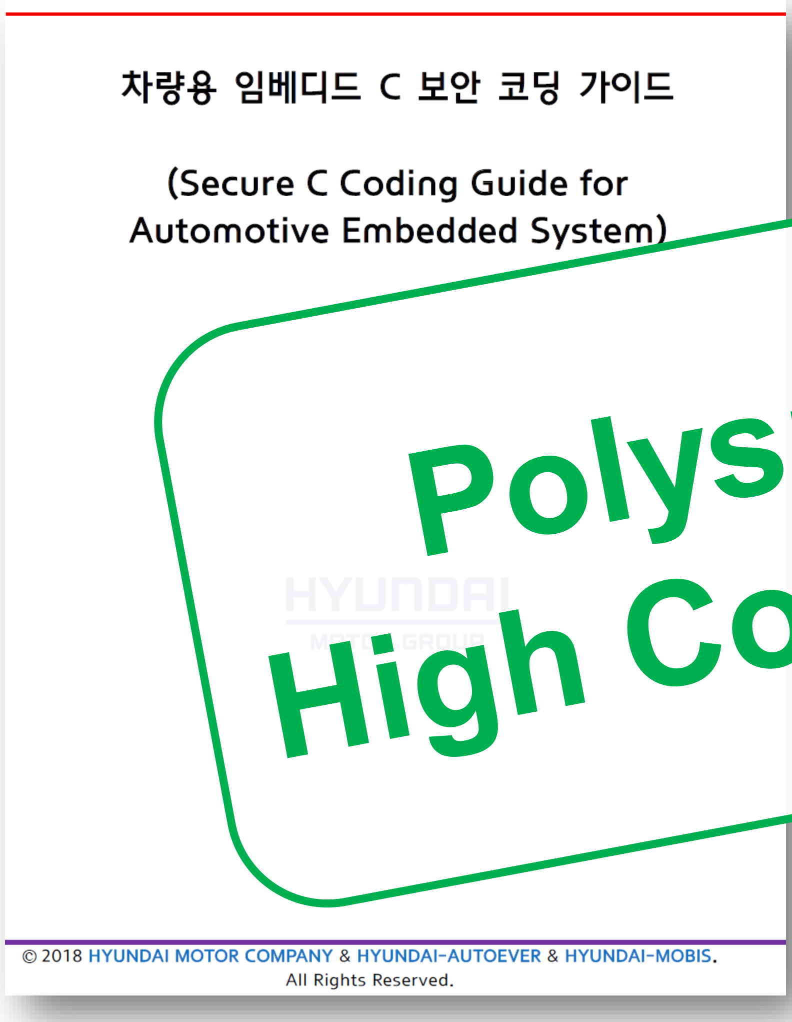
can be used as the basis for evidencing and evaluating …

6. Link with ISO/SAE DIS 21434

Paragraph	Clauses from ISO/SAE DIS 21434
7.2.2.1. The vehicle manufacturer shall demonstrate to an Approval Authority or Technical Service that their Cyber Security Management System applies to the following phases:	
Development phase	Clauses 9, 10, 11, 15
Production phase	Clause 12
Post-production phase	Clauses 7, 13, 14, 15
…………	

New Cybersecurity Requirements for Automotive in Korea

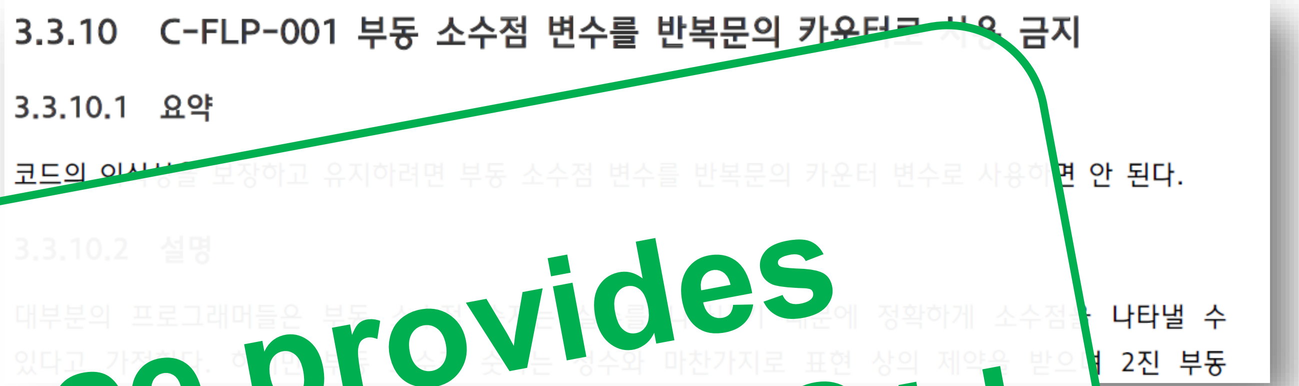
Secure Coding Guide for Automotive Embedded System



차량용 임베디드 C 보안 코딩 가이드

(Secure C Coding Guide for Automotive Embedded System)

© 2018 HYUNDAI MOTOR COMPANY & HYUNDAI-AUTOEVER & HYUNDAI-MOBIS.
All Rights Reserved.



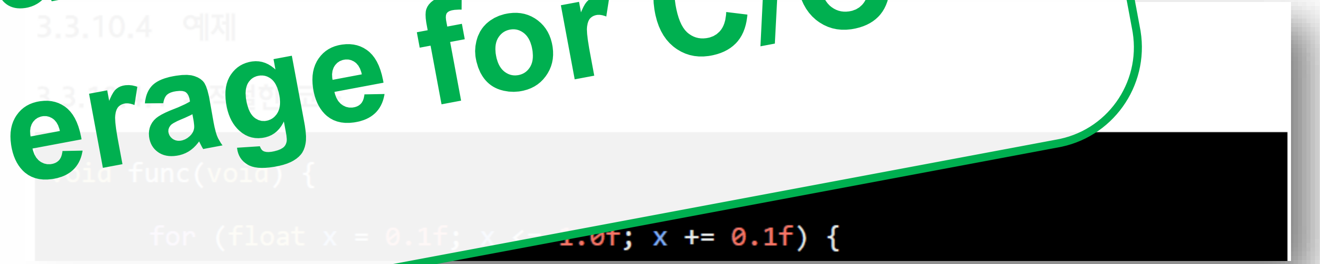
3.3.10 C-FLP-001 부동 소수점 변수를 반복문의 카운터로 사용 금지

3.3.10.1 요약

코드의 이식성을 보장하고 유지하려면 부동 소수점 변수를 반복문의 카운터 변수로 사용하면 안 된다.

3.3.10.2 설명

대부분의 프로그래머들은 부동 소수점 숫자는 실수를 [illegible] 때문에 정확하게 소수점을 나타낼 수 있다고 가정한다. 하지만 부동 소수점 숫자는 정수와 마찬가지로 표현 상의 제약을 받으며 2진 부동

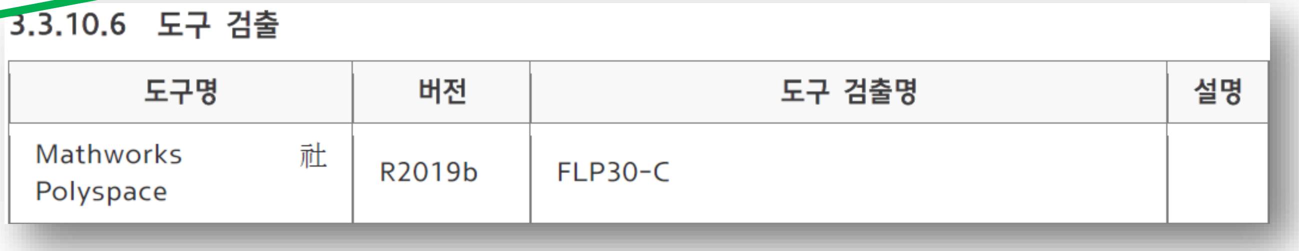


3.3.10.4 예제

3.3.10.5 부적합한 코드

```
void func(void) {
    for (float x = 0.1f; x <= 1.0f; x += 0.1f) {
```

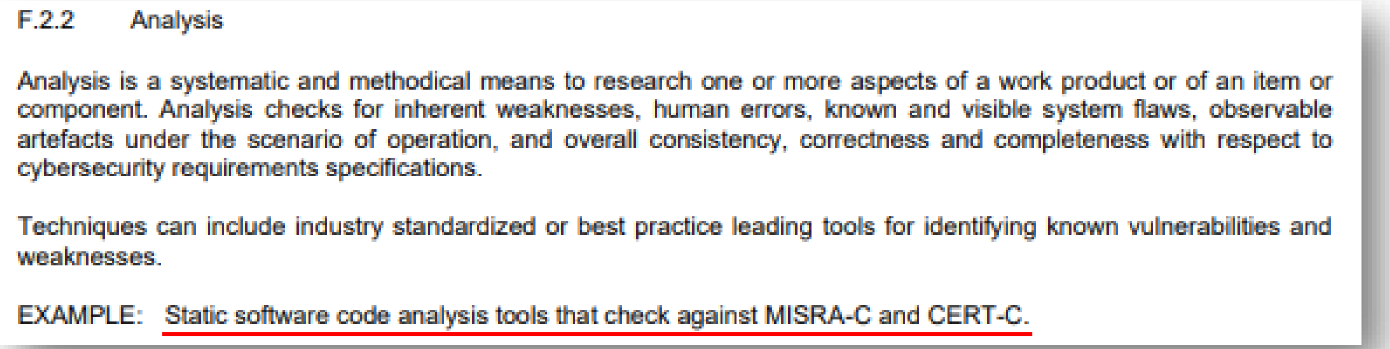
Polyspace provides High Coverage for C/C++



3.3.10.6 도구 검출

도구명	버전	도구 검출명	설명
Mathworks 社 Polyspace	R2019b	FLP30-C	

Automotive Cybersecurity
&
Static Application Security Testing



F.2.2 Analysis

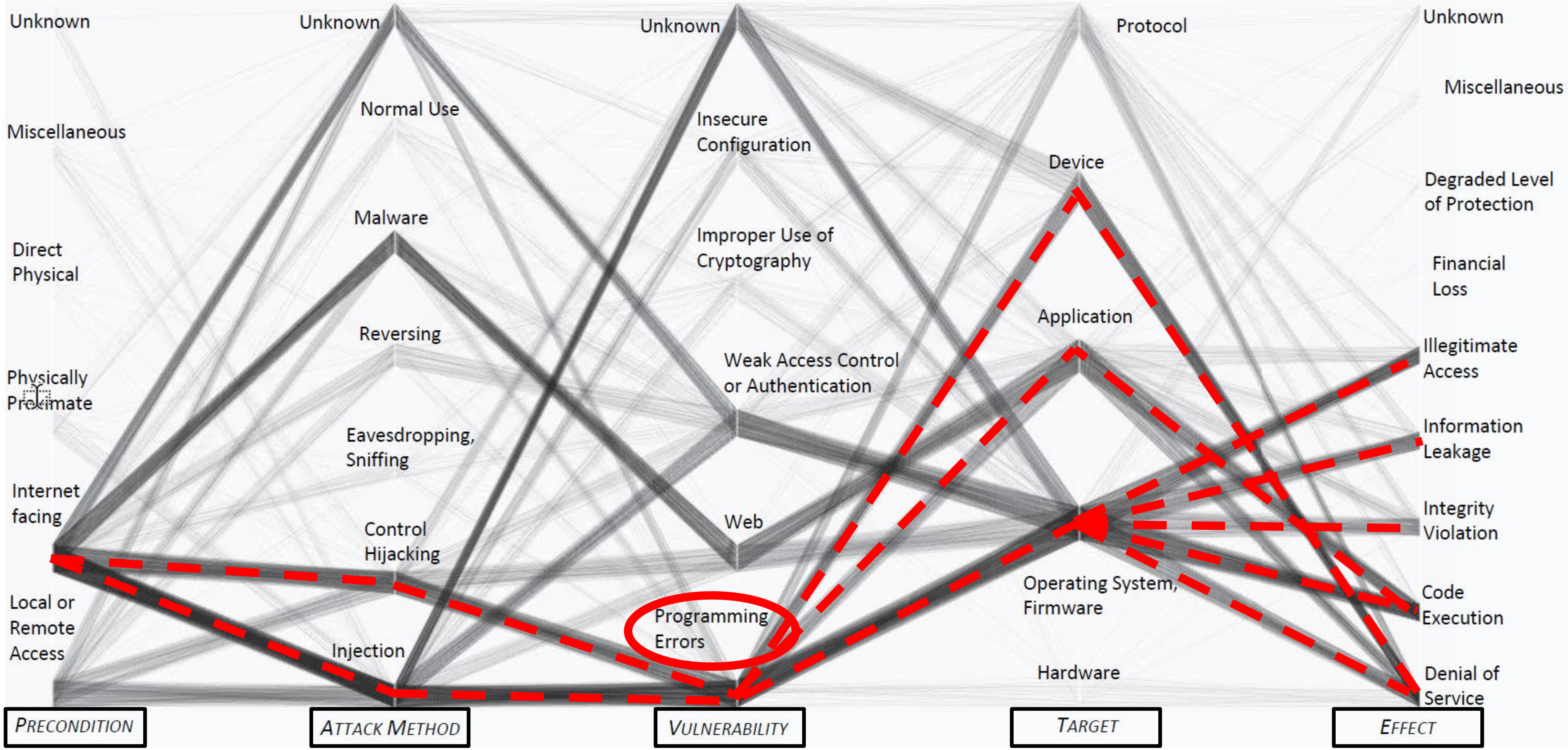
Analysis is a systematic and methodical means to research one or more aspects of a work product or of an item or component. Analysis checks for inherent weaknesses, human errors, known and visible system flaws, observable artefacts under the scenario of operation, and overall consistency, correctness and completeness with respect to cybersecurity requirements specifications.

Techniques can include industry standardized or best practice leading tools for identifying known vulnerabilities and weaknesses.

EXAMPLE: Static software code analysis tools that check against MISRA-C and CERT-C.

From : ISO/SAE DIS 21434

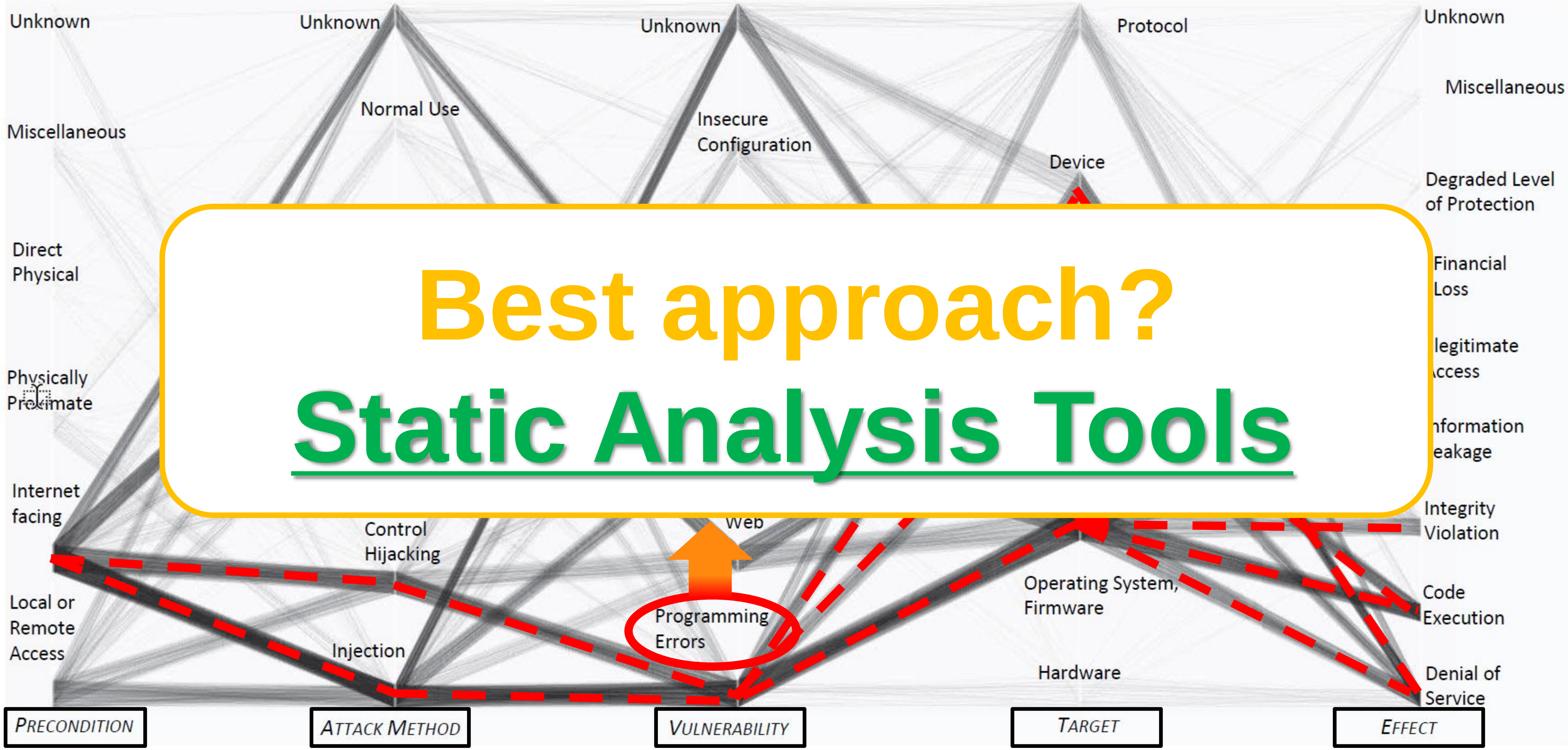
Common Cyberattack Scenarios



Programming errors are one major source of vulnerabilities

Source: *Embedded Systems Security*, D. Papp et al, IEEE Conf. Sec. Privacy & Trust, 2015.

Common Cyberattack Scenarios



Best approach?

Static Analysis Tools

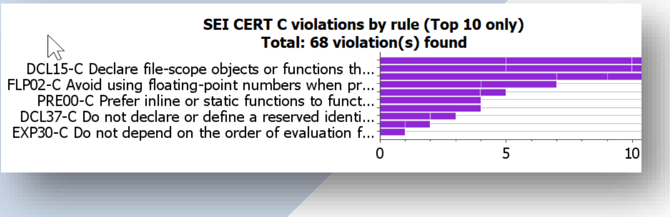
Programming errors are one major source of vulnerabilities

Source: *Embedded Systems Security*, D. Papp et al, IEEE Conf. Sec. Privacy & Trust, 2015.

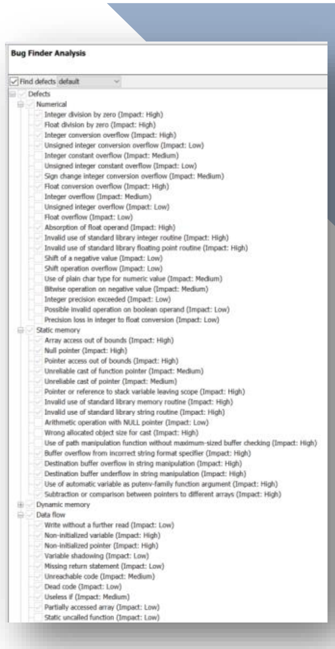
Static Application Security “Testing” (SAST) with Polyspace

Analysis & proof instead of dynamic execution

1. Enforce Secure Coding Guidelines


SEI CERT C violations by rule (Top 10 only)
Total: 68 violation(s) found

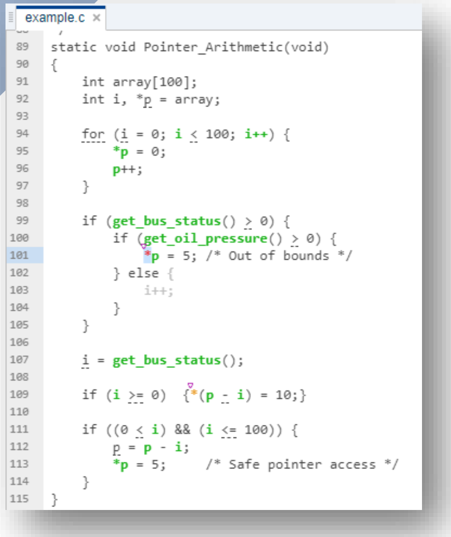
2. Detect Security Flaws


Bug Finder Analysis

☑ Find defects default

- ☑ Defects
  - ☑ Numerical
    - ☑ Integer division by zero (Impact: High)
    - ☑ Float division by zero (Impact: High)
    - ☑ Integer conversion overflow (Impact: High)
    - ☑ Unsigned integer conversion overflow (Impact: Low)
    - ☐ Integer constant overflow (Impact: Medium)
    - ☐ Unsigned integer constant overflow (Impact: Low)
    - ☑ Sign change integer conversion overflow (Impact: Medium)
    - ☑ Float conversion overflow (Impact: High)
    - ☐ Integer overflow (Impact: Medium)
    - ☐ Unsigned integer overflow (Impact: Low)
    - ☐ Float overflow (Impact: Low)
    - ☑ Absorption of float operand (Impact: High)
    - ☑ Invalid use of standard library integer routine (Impact: High)
    - ☑ Invalid use of standard library floating point routine (Impact: High)
    - ☐ Shift of a negative value (Impact: Low)
    - ☐ Shift operation overflow (Impact: Low)
    - ☐ Use of plain char type for numeric value (Impact: Medium)
    - ☐ Bitwise operation on negative value (Impact: Medium)
    - ☐ Integer precision exceeded (Impact: Low)
    - ☐ Possible invalid operation on boolean operand (Impact: Low)
    - ☐ Precision loss in integer to float conversion (Impact: Low)
  - ☑ Static memory
    - ☑ Array access out of bounds (Impact: High)
    - ☑ Null pointer (Impact: High)
    - ☑ Pointer access out of bounds (Impact: High)
    - ☑ Unreliable cast of function pointer (Impact: Medium)
    - ☑ Unreliable cast of pointer (Impact: Medium)
    - ☑ Pointer or reference to stack variable leaving scope (Impact: High)
    - ☑ Invalid use of standard library memory routine (Impact: High)
    - ☑ Invalid use of standard library string routine (Impact: High)
    - ☐ Arithmetic operation with NULL pointer (Impact: Low)
    - ☐ Wrong allocated object size for cast (Impact: High)
    - ☐ Use of path manipulation function without maximum-sized buffer checking (Impact: High)
    - ☑ Buffer overflow from incorrect string format specifier (Impact: High)
    - ☑ Destination buffer overflow in string manipulation (Impact: High)
    - ☑ Destination buffer underflow in string manipulation (Impact: High)
    - ☑ Use of automatic variable as putenv-family function argument (Impact: High)
    - ☑ Subtraction or comparison between pointers to different arrays (Impact: High)
  - ☑ Dynamic memory
  - ☑ Data flow
    - ☑ Write without a further read (Impact: Low)
    - ☑ Non-initialized variable (Impact: High)
    - ☑ Non-initialized pointer (Impact: High)
    - ☑ Variable shadowing (Impact: Low)
    - ☑ Missing return statement (Impact: Low)
    - ☑ Unreachable code (Impact: Medium)
    - ☑ Dead code (Impact: Low)
    - ☑ Useless if (Impact: Medium)
    - ☑ Partially accessed array (Impact: Low)
    - ☐ Static uncalled function (Impact: Low)

3. Prove Absence of Critical Vulnerabilities

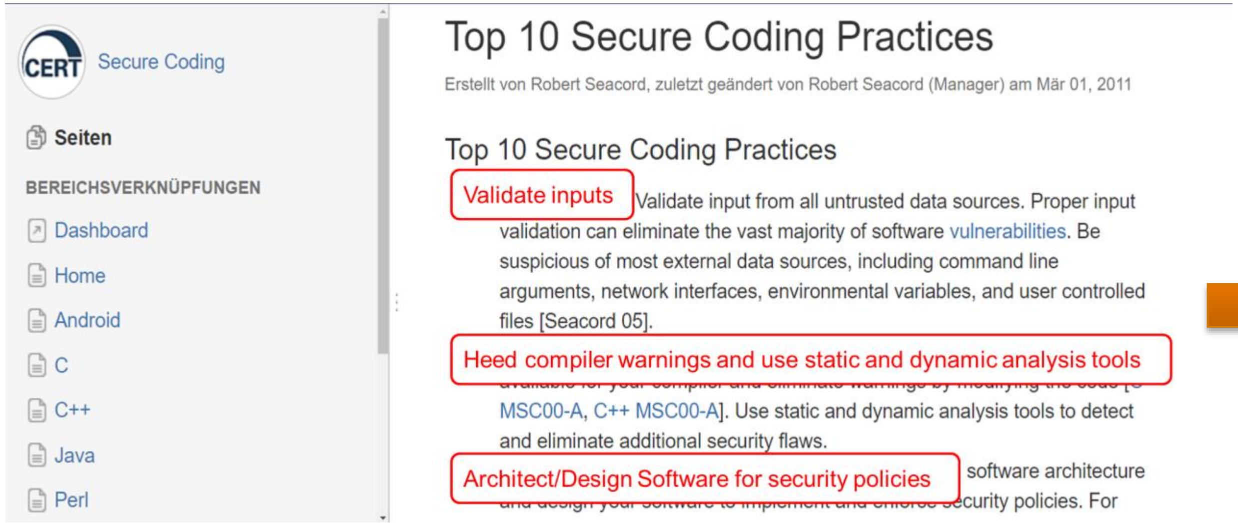


```c
static void Pointer_Arithmetic(void)
{
    int array[100];
    int i, *p = array;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5; /* Out of bounds */
        } else {
            i++;
        }
    }

    i = get_bus_status();

    if (i >= 0)  {*(p - i) = 10;}

    if ((0 < i) && (i <= 100)) {
        p = p - i;
        *p = 5;     /* Safe pointer access */
    }
}
```

1. Enforce Secure Coding Guidelines

CERT C(++) Secure Coding Standard in Polyspace

- Coding standard to improve safety, reliability and security
- Cross-referenced by MISRA, CWE and others



Top 10 Secure Coding Practices

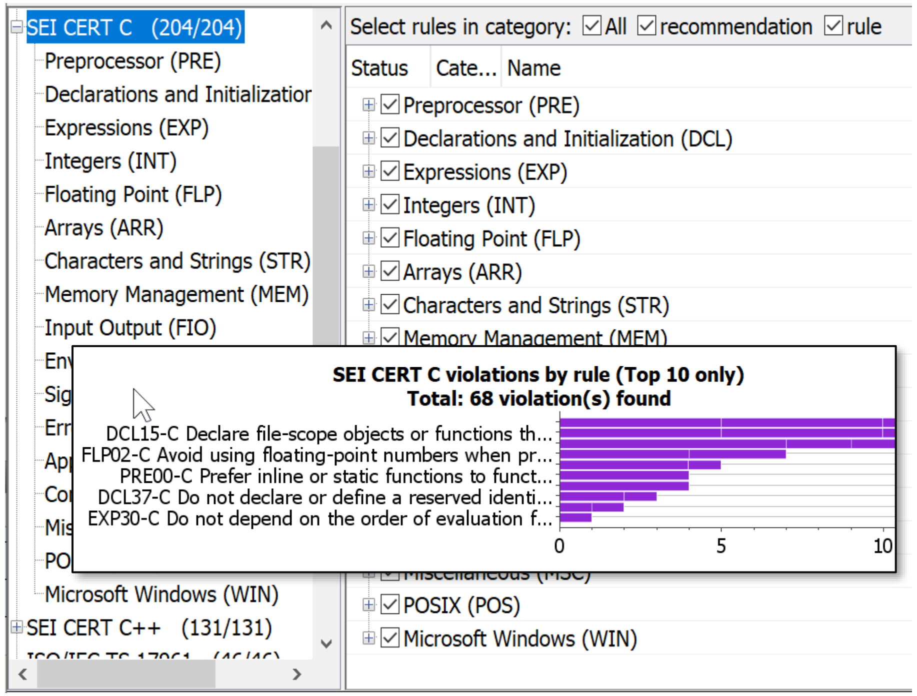
Erstellt von Robert Seacord, zuletzt geändert von Robert Seacord (Manager) am Mär 01, 2011

Top 10 Secure Coding Practices

Validate inputs Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].

Heed compiler warnings and use static and dynamic analysis tools MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws.

Architect/Design Software for security policies software architecture ecurity policies. For


SEI CERT C (204/204)
Preprocessor (PRE)
Declarations and Initializatior
Expressions (EXP)
Integers (INT)
Floating Point (FLP)
Arrays (ARR)
Characters and Strings (STR)
Memory Management (MEM)
Input Output (FIO)
Env
Sig
Err
Ap
Co
Mis
PO
Microsoft Windows (WIN)
SEI CERT C++ (131/131)
Select rules in category: ☑ All ☑ recommendation ☑ rule
Status
Cate...
Name
☑ Preprocessor (PRE)
☑ Declarations and Initialization (DCL)
☑ Expressions (EXP)
☑ Integers (INT)
☑ Floating Point (FLP)
☑ Arrays (ARR)
☑ Characters and Strings (STR)
☑ Memory Management (MEM)
☑ POSIX (POS)
☑ Microsoft Windows (WIN)
SEI CERT C violations by rule (Top 10 only)
Total: 68 violation(s) found
DCL15-C Declare file-scope objects or functions th...
FLP02-C Avoid using floating-point numbers when pr...
PRE00-C Prefer inline or static functions to funct...
DCL37-C Do not declare or define a reserved identi...
EXP30-C Do not depend on the order of evaluation f...
0
5
10

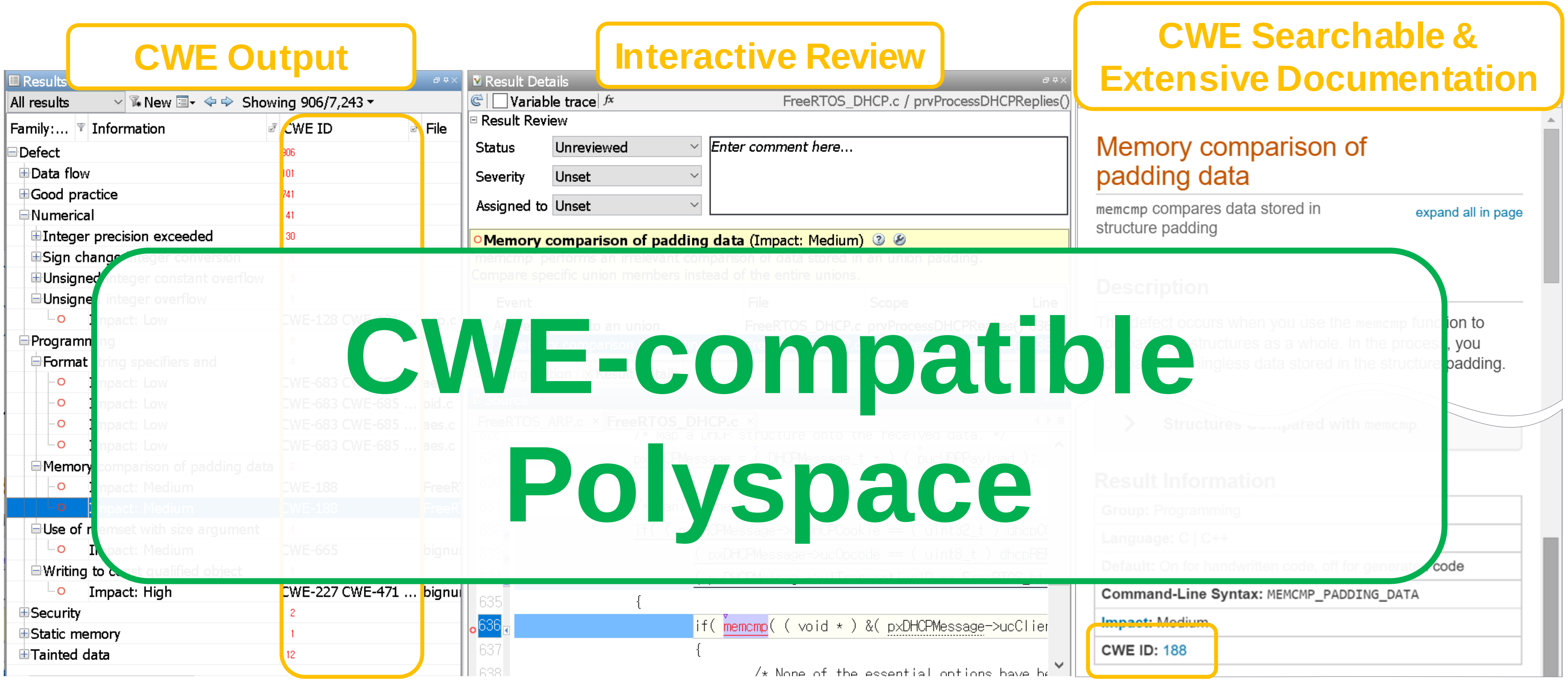
Polyspace has 100% coverage of automatable rules

Other security-relevant coding standards in Polyspace: MISRA, ISO/IEC TS 17961

2. Detect Security Flaws

Common Weakness Enumeration (CWE) with Polyspace

- MITRE categorizes to stop/eliminate those known programming errors before production
- Polyspace provides CWE mappings & views for C and C++



CWE Output

Interactive Review

CWE Searchable & Extensive Documentation

CWE-compatible Polyspace

Results

All results | New | Showing 906/7,243

Family:...	Information	CWE ID	File
⊟ Defect		906	
⊞ Data flow		101	
⊞ Good practice		741	
⊟ Numerical		41	
⊞ Integer precision exceeded		30	
⊞ Sign change integer conversion		1	
⊞ Unsigned integer constant overflow		9	
⊟ Unsigned integer overflow		1	
○	Impact: Low	CWE-128 CWE-191 ...	p.c
⊟ Programming		8	
⊟ Format string specifiers and		4	
○	Impact: Low	CWE-683 CWE-685 ...	aes.c
○	Impact: Low	CWE-683 CWE-685 ...	oid.c
○	Impact: Low	CWE-683 CWE-685 ...	aes.c
○	Impact: Low	CWE-683 CWE-685 ...	aes.c
⊟ Memory comparison of padding data		2	
○	Impact: Medium	CWE-188	FreeR
○	Impact: Medium	CWE-188	FreeR
⊟ Use of memset with size argument		1	
○	Impact: Medium	CWE-665	bignu
⊟ Writing to const qualified object		1	
○	Impact: High	CWE-227 CWE-471 ...	bignu
⊞ Security		2	
⊞ Static memory		1	
⊞ Tainted data		12	

Result Details

□ Variable trace | FreeRTOS_DHCP.c / prvProcessDHCPReplies()

⊟ Result Review

Status: Unreviewed
Severity: Unset
Assigned to: Unset
Enter comment here...

○ **Memory comparison of padding data** (Impact: Medium)

memcmp performs an irrelevant comparison of data stored in an union padding.
Compare specific union members instead of the entire unions.

Event	File	Scope	Line
Assignment to an union	FreeRTOS_DHCP.c	prvProcessDHCPReplies()	636
Memory comparison of padding data	FreeRTOS_DHCP.c	prvProcessDHCPReplies()	636

Configuration | Result Details

Source

FreeRTOS_ARP.c × | FreeRTOS_DHCP.c ×

```
/* Map a DHCP structure onto the received data. */
pxDHCPMessage = ( DHCPMessage_t * ) ( pucUDPPayload );

/* Sanity check. */
if( ( pxDHCPMessage->ulDHCPCookie == ( uint32_t ) dhcpC
    ( pxDHCPMessage->ucOpcode == ( uint8_t ) dhcpREF
    ( pxDHCPMessage->ulTransactionID == FreeRTOS_htc
{
if( memcmp( ( void * ) &( pxDHCPMessage->ucClien
{
    /* None of the essential options have be
```

Memory comparison of padding data

`memcmp` compares data stored in structure padding

expand all in page

Description

This defect occurs when you use the `memcmp` function to compare structures as a whole. In the process, you compare meaningless data stored in the structure padding.

Structures compared with `memcmp`

Result Information

Group: Programming

Language: C | C++

Default: On for handwritten code, off for generated code

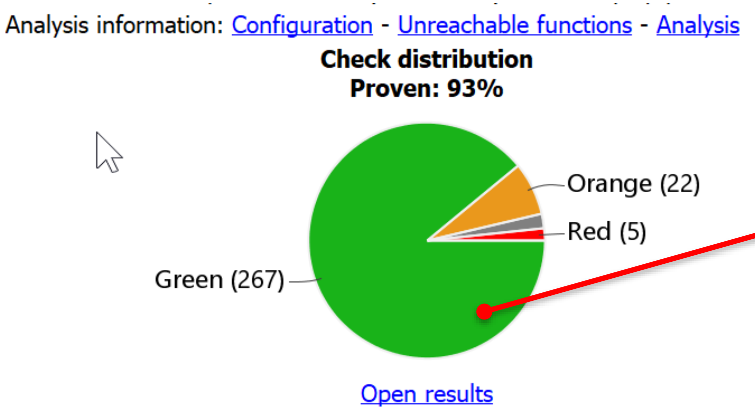
Command-Line Syntax: `MEMCMP_PADDING_DATA`

Impact: Medium

CWE ID: 188

3. Prove Absence of Critical Vulnerabilities

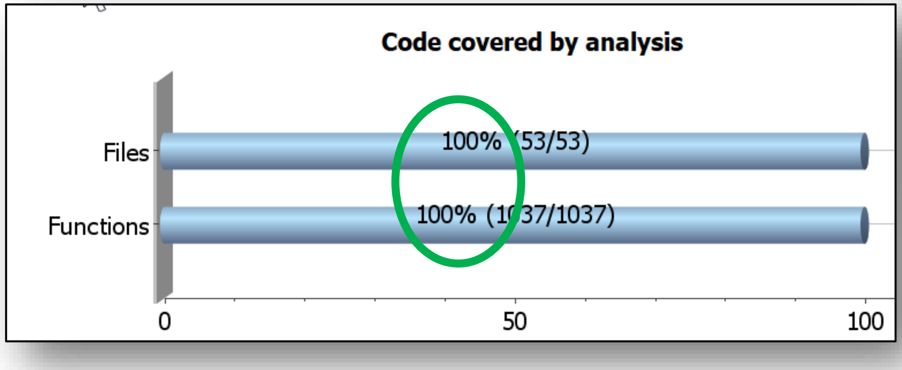
Polyspace
Code Prover



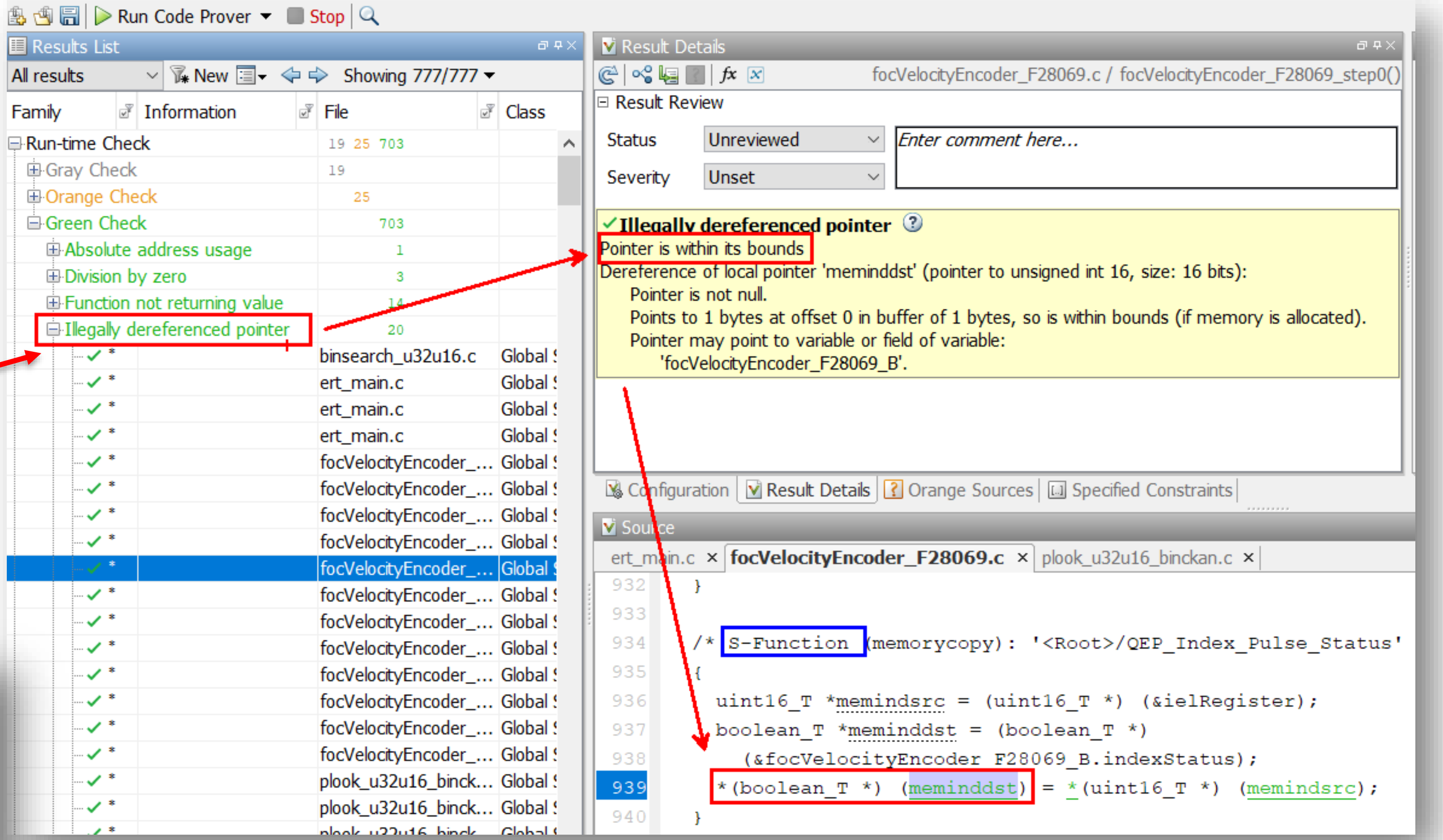
Analysis information: Configuration - Unreachable functions - Analysis

Check distribution
Proven: 93%

Open results



Code covered by analysis



Family	Information	File	Class
Run-time Check		19 25 703	
Gray Check		19	
Orange Check		25	
Green Check		703	
Absolute address usage		1	
Division by zero		3	
Function not returning value		14	
Illegally dereferenced pointer		20	
✓ *		binsearch_u32u16.c	Global
✓ *		ert_main.c	Global
✓ *		ert_main.c	Global
✓ *		ert_main.c	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		focVelocityEncoder_…	Global
✓ *		plook_u32u16_binck…	Global
✓ *		plook_u32u16_binck…	Global

Result Details
focVelocityEncoder_F28069.c / focVelocityEncoder_F28069_step0()
Result Review

Status: Unreviewed
Severity: Unset
Enter comment here...

✓ **Illegally dereferenced pointer**
Pointer is within its bounds
Dereference of local pointer 'meminddst' (pointer to unsigned int 16, size: 16 bits):
Pointer is not null.
Points to 1 bytes at offset 0 in buffer of 1 bytes, so is within bounds (if memory is allocated).
Pointer may point to variable or field of variable:
'focVelocityEncoder_F28069_B'.

```
}

/* S-Function (memorycopy): '<Root>/QEP_Index_Pulse_Status'
{
  uint16_T *memindsrc = (uint16_T *) (&ielRegister);
  boolean_T *meminddst = (boolean_T *)
    (&focVelocityEncoder_F28069_B.indexStatus);
  *(boolean_T *) (meminddst) = *(uint16_T *) (memindsrc);
}
```

Considers *all* inputs & *all* program states

Static Code Analysis as Recommended Method in ISO 21434

Table E.4 - Methods for verification of integration ([RQ-10-12])

Topic	CAL			
	1	2	3	4
Requirement-based test	✓	✓	✓	✓
Interface test	✓	✓	✓	✓
Resource usage evaluation	✓	✓	✓	✓
Verification of the control flow and data flow			✓	✓
Static code analysis	✓	✓	✓	✓

Table E.5 - Methods for deriving test cases ([RQ-10-14])

Topic	CAL			
	1	2	3	4
Analysis of requirements	✓	✓	✓	✓
Generation and analysis of equivalence classes			✓	✓
Boundary values analysis			✓	✓
Error guessing based on knowledge or experience				

Table E.9 - Topic list ([RQ-10-20])

Topic	CAL			
	1	2	3	4
Use of language subsets	✓	✓	✓	✓
Enforcement of strong typing	✓	✓	✓	✓
Use of defensive implementation techniques			✓	✓

Polyspace Bug Finder
Polyspace Code Prover

Catching Up with Cybersecurity in Three Steps

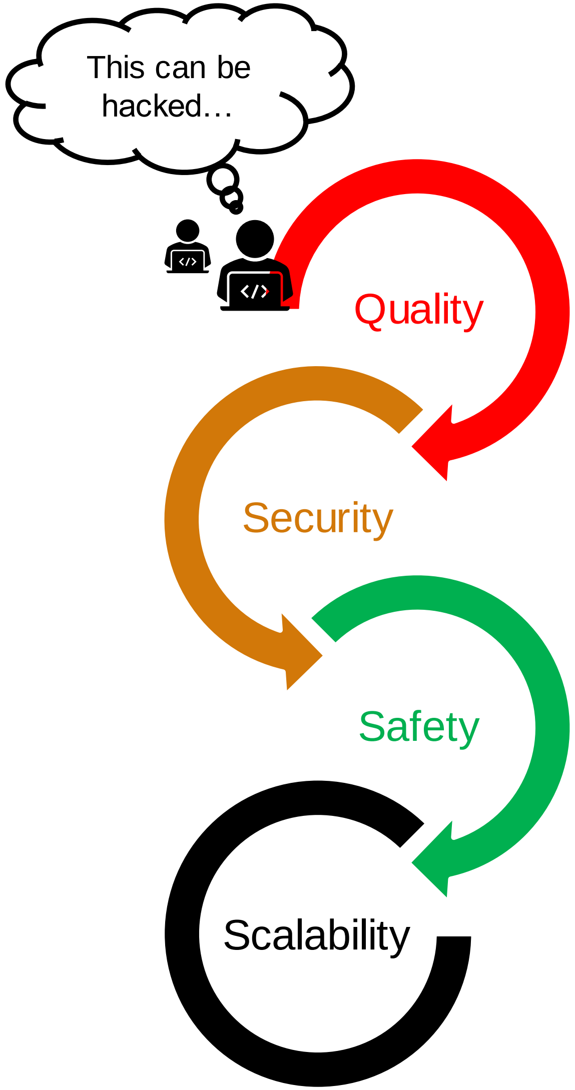
https://kr.mathworks.com/content/dam/mathworks/conference-or-academic-paper/increasing-resilience-to-cyberattacks-through-advanced-use-of-static-code-analysis.pdf

Catching up with Cybersecurity in three steps:

1. Train developers…
   - Best practices & coding guidelines to avoid common errors
   - Distribute workload on the many, “shift left”

2. Miss “no” defects with static analysis…
   - *Sound* analysis is superior to Fuzz Testing
   - Considers all corner cases, guaranteed robustness

3. Automate, Collaborate & Monitor…
   - Rigorous “nightly security reviews” without experts
   - Supporting security code reviews
   - Quality gates to keep your software robust & clean

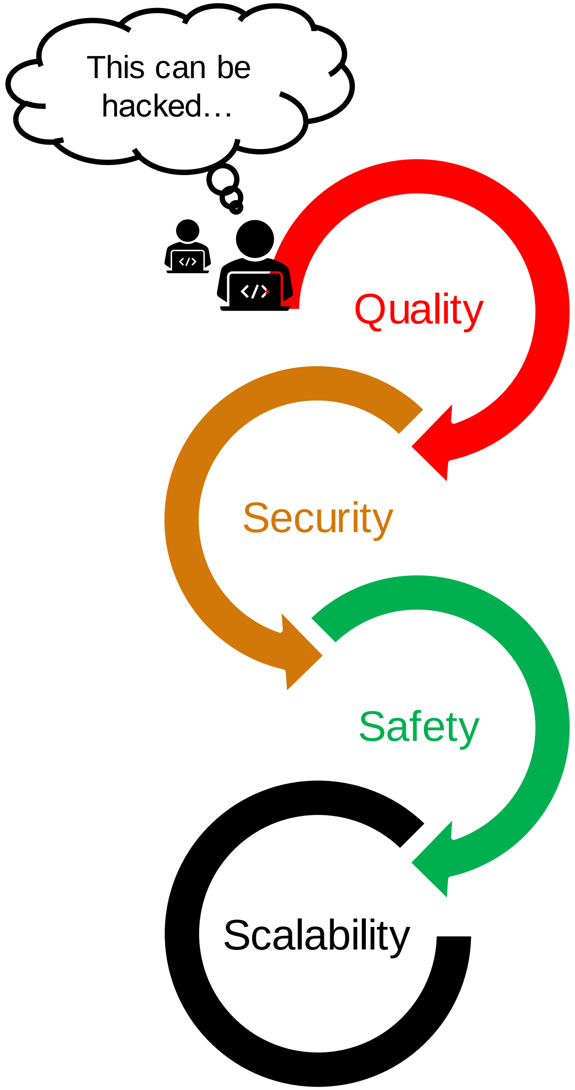


Catching up with Cybersecurity in three steps:

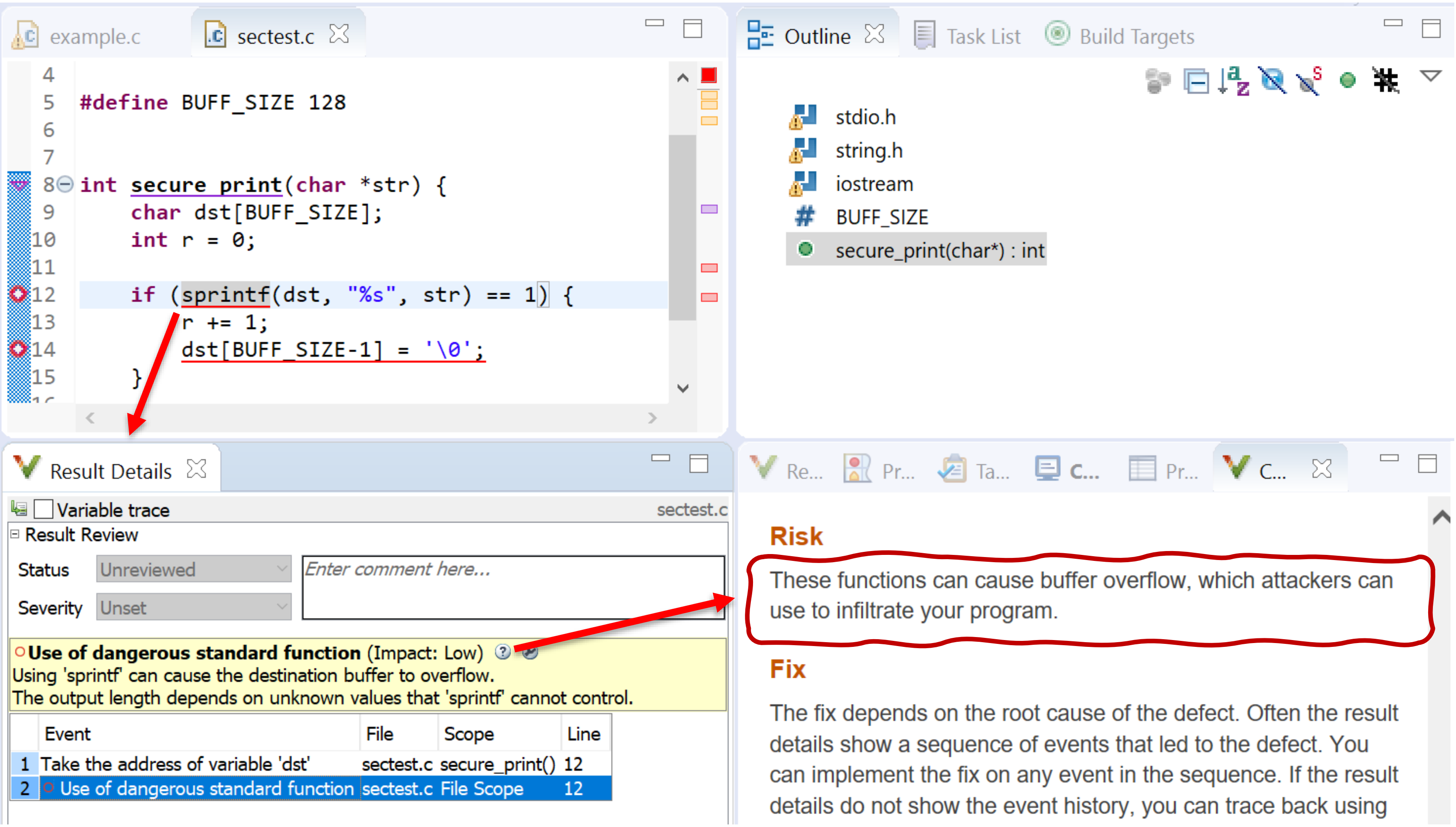
➡ Train developers…
   - Best practices & coding guidelines to avoid common errors
   - Distribute workload on the many, “shift left”

2. Miss “no” defects with static analysis…
   - *Sound* analysis is superior to Fuzz Testing
   - Considers all corner cases, guaranteed robustness

3. Automate, Collaborate & Monitor…
   - Rigorous “nightly security reviews” without experts
   - Central result storage & review
   - Quality gates to keep your software robust & clean



Follow Secure Coding Guidelines and Practices As You Code



```
#define BUFF_SIZE 128


int secure_print(char *str) {
     char dst[BUFF_SIZE];
     int r = 0;

     if (sprintf(dst, "%s", str) == 1) {
         r += 1;
         dst[BUFF_SIZE-1] = '\0';
     }
```

- stdio.h
- string.h
- iostream
- BUFF_SIZE
- secure_print(char*) : int

□ Variable trace — sectest.c

Result Review

Status: Unreviewed

Severity: Unset

Enter comment here...

Use of dangerous standard function (Impact: Low)
Using 'sprintf' can cause the destination buffer to overflow.
The output length depends on unknown values that 'sprintf' cannot control.

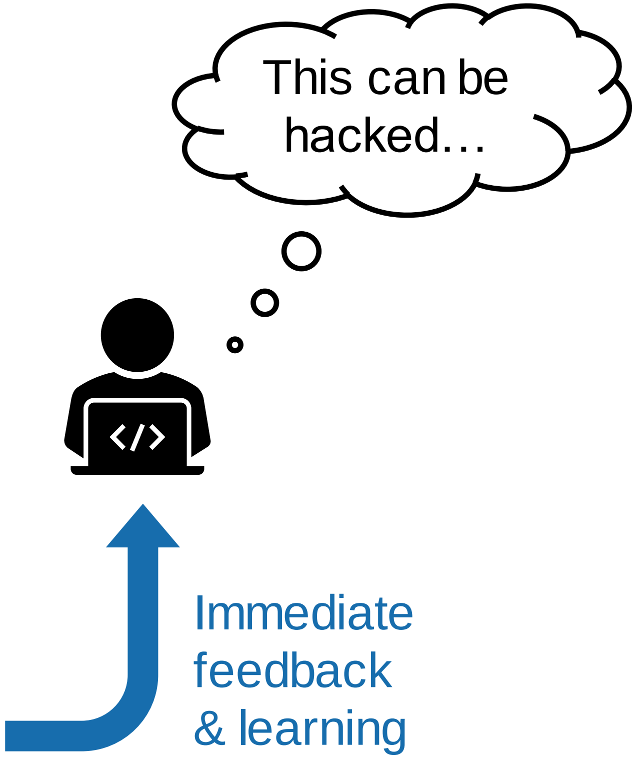
	Event	File	Scope	Line
1	Take the address of variable 'dst'	sectest.c	secure_print()	12
2	Use of dangerous standard function	sectest.c	File Scope	12

Risk

These functions can cause buffer overflow, which attackers can use to infiltrate your program.

Fix

The fix depends on the root cause of the defect. Often the result details show a sequence of events that led to the defect. You can implement the fix on any event in the sequence. If the result details do not show the event history, you can trace back using



Immediate feedback & learning

CERT CWE MISRA

Polyspace has 99.4% coverage of secure coding guideline CERT-C(++), identifies common programming errors (CWE) and computes complexity metrics

Fixing Flaws Requires Understanding

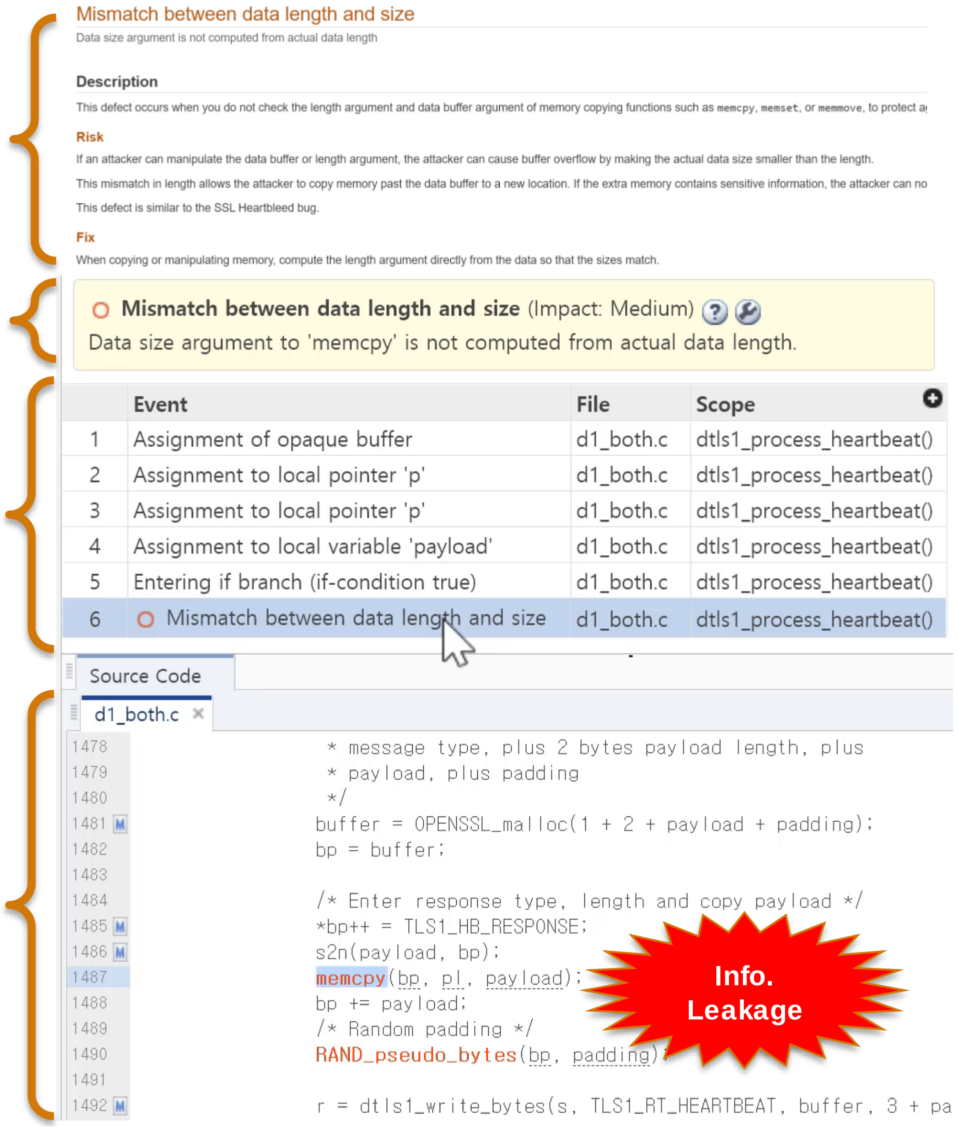
Root cause analysis & attack path analysis made easy

- I don’t understand the tool warning…
- …suppress/ignore ➔ missed vulnerabilities

Event traces:

1. Ease comprehension
   - Control decisions to reach vulnerability
2. Support root cause & attack path analysis
   - Partial attack path for free
3. Shorten debugging time
   - No reconstruction in debugger needed

Interactive review interfaces reduce oversight



Extensive Documentation

Mismatch between data length and size

Data size argument is not computed from actual data length

Description

This defect occurs when you do not check the length argument and data buffer argument of memory copying functions such as `memcpy`, `memset`, or `memmove`, to protect a

Risk

If an attacker can manipulate the data buffer or length argument, the attacker can cause buffer overflow by making the actual data size smaller than the length.

This mismatch in length allows the attacker to copy memory past the data buffer to a new location. If the extra memory contains sensitive information, the attacker can no

This defect is similar to the SSL Heartbleed bug.

Fix

When copying or manipulating memory, compute the length argument directly from the data so that the sizes match.

Defect description

○ **Mismatch between data length and size** (Impact: Medium)

Data size argument to 'memcpy' is not computed from actual data length.

Interactive event trace

	Event	File	Scope
1	Assignment of opaque buffer	d1_both.c	dtls1_process_heartbeat()
2	Assignment to local pointer 'p'	d1_both.c	dtls1_process_heartbeat()
3	Assignment to local pointer 'p'	d1_both.c	dtls1_process_heartbeat()
4	Assignment to local variable 'payload'	d1_both.c	dtls1_process_heartbeat()
5	Entering if branch (if-condition true)	d1_both.c	dtls1_process_heartbeat()
6	○ Mismatch between data length and size	d1_both.c	dtls1_process_heartbeat()

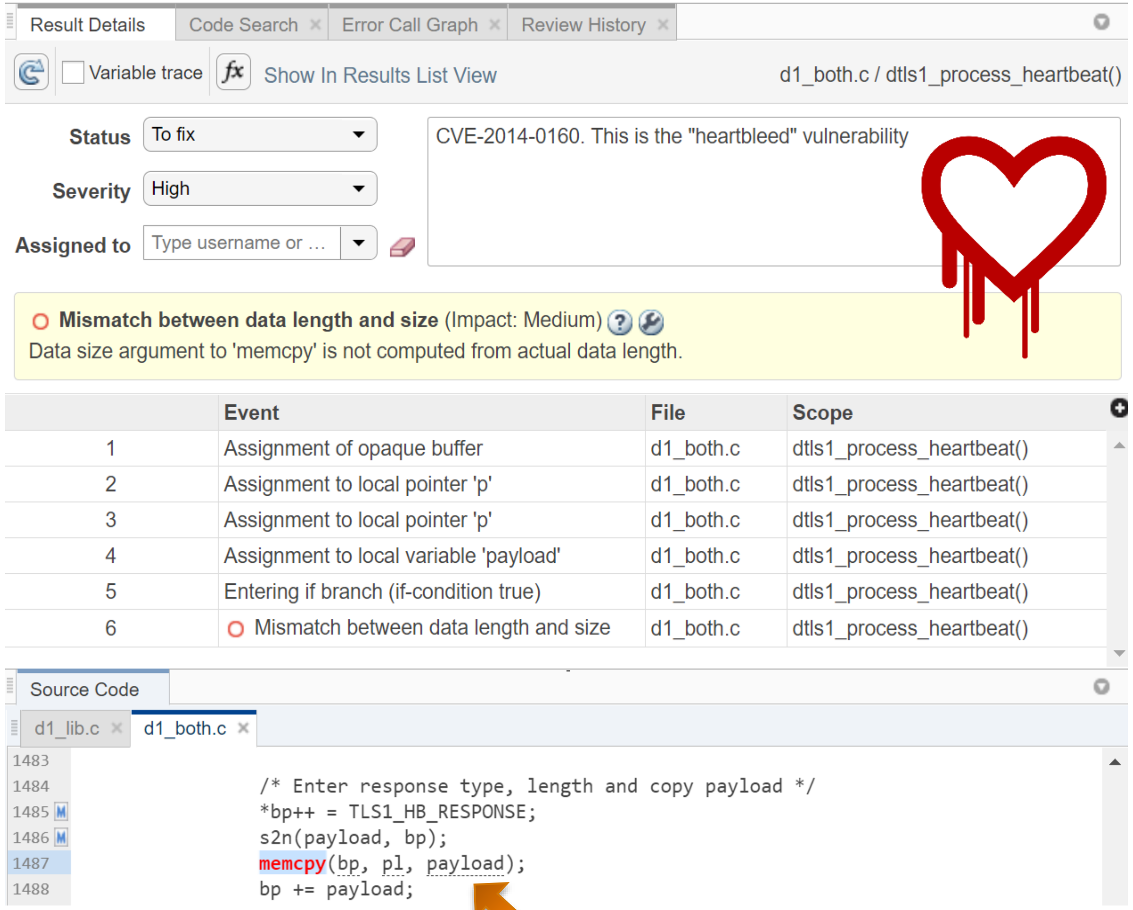
Source with debug information

```
 * message type, plus 2 bytes payload length, plus
 * payload, plus padding
 */
buffer = OPENSSL_malloc(1 + 2 + payload + padding);
bp = buffer;

/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
/* Random padding */
RAND_pseudo_bytes(bp, padding);

r = dtls1_write_bytes(s, TLS1_RT_HEARTBEAT, buffer, 3 + pa
```

Info. Leakage

Beyond Guidelines: Dedicated Security Checkers

Examples: OpenSSL Heartbleed (lacking data dependency), Jeep Hack (weak RNG)



d1_both.c / dtls1_process_heartbeat()

Status: To fix

Severity: High

Assigned to: Type username or ...

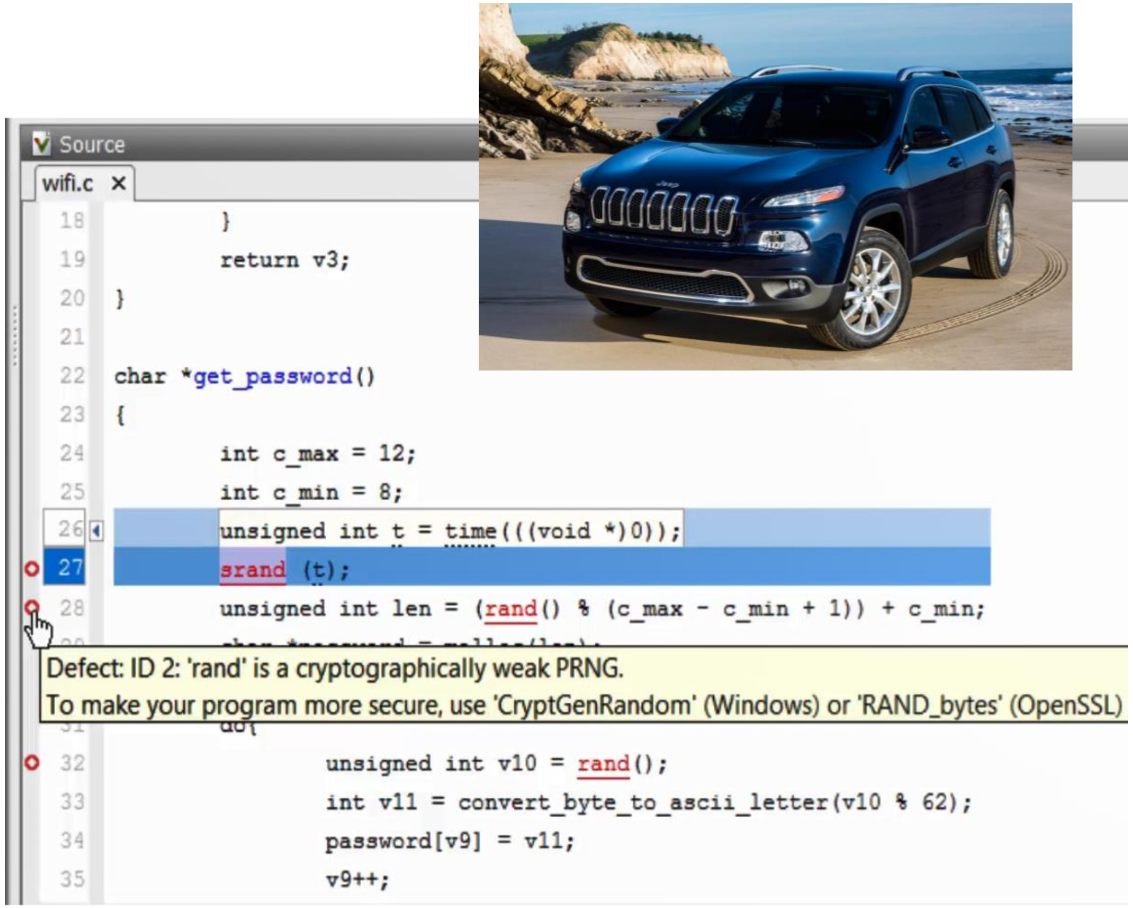
CVE-2014-0160. This is the "heartbleed" vulnerability

○ **Mismatch between data length and size** (Impact: Medium)
Data size argument to 'memcpy' is not computed from actual data length.

	Event	File	Scope
1	Assignment of opaque buffer	d1_both.c	dtls1_process_heartbeat()
2	Assignment to local pointer 'p'	d1_both.c	dtls1_process_heartbeat()
3	Assignment to local pointer 'p'	d1_both.c	dtls1_process_heartbeat()
4	Assignment to local variable 'payload'	d1_both.c	dtls1_process_heartbeat()
5	Entering if branch (if-condition true)	d1_both.c	dtls1_process_heartbeat()
6	○ Mismatch between data length and size	d1_both.c	dtls1_process_heartbeat()

```
/* Enter response type, length and copy payload */
*bp++ = TLS1_HB_RESPONSE;
s2n(payload, bp);
memcpy(bp, pl, payload);
bp += payload;
```

Wrong variable for length



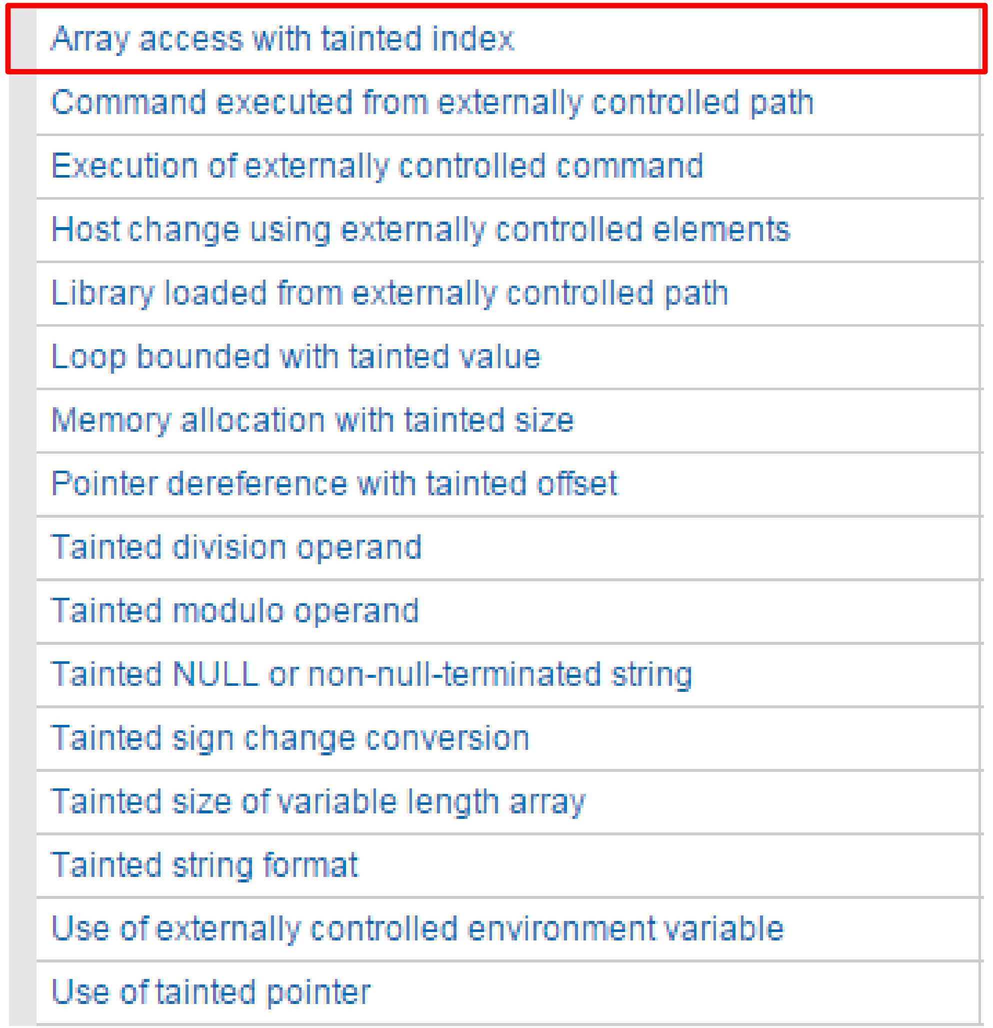
```
        }
        return v3;
}

char *get_password()
{
        int c_max = 12;
        int c_min = 8;
        unsigned int t = time(((void *)0));
        srand (t);
        unsigned int len = (rand() % (c_max - c_min + 1)) + c_min;
        char *password = malloc(len);
```

Defect: ID 2: 'rand' is a cryptographically weak PRNG.
To make your program more secure, use 'CryptGenRandom' (Windows) or 'RAND_bytes' (OpenSSL)

```
        do{
                unsigned int v10 = rand();
                int v11 = convert_byte_to_ascii_letter(v10 % 62);
                password[v9] = v11;
                v9++;
```

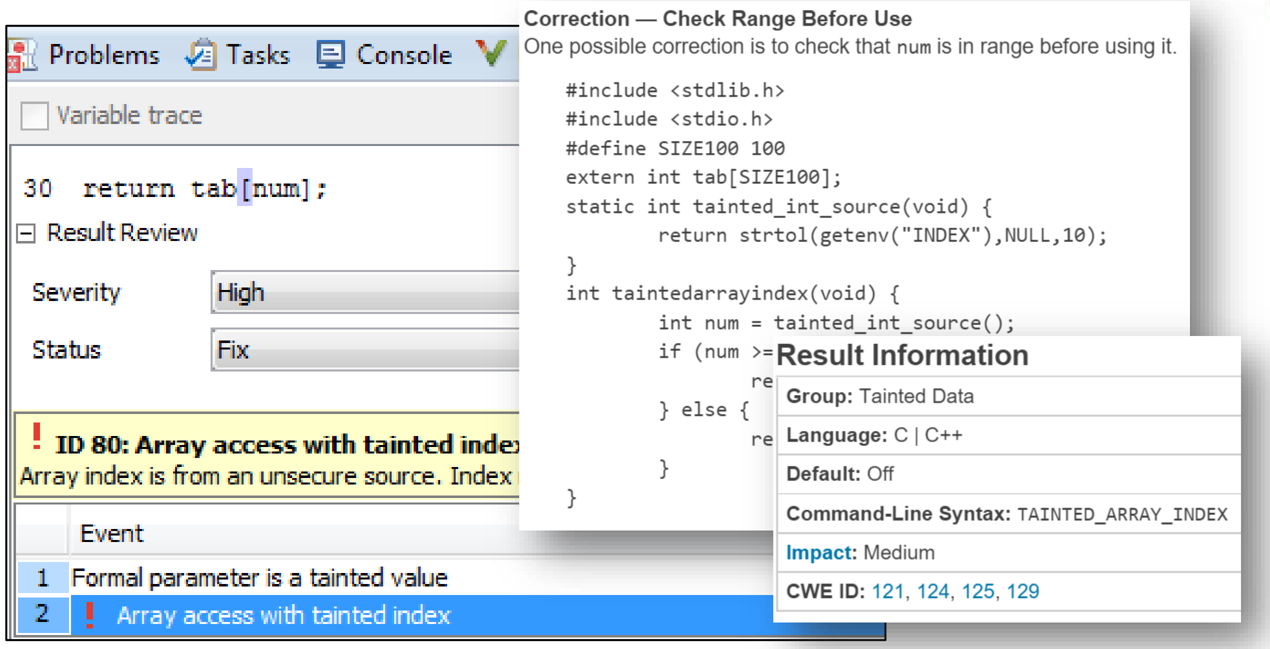
Beyond Guidelines: Automated Taint Analysis

Defects related to data from an unsecure source



- Array access with tainted index
- Command executed from externally controlled path
- Execution of externally controlled command
- Host change using externally controlled elements
- Library loaded from externally controlled path
- Loop bounded with tainted value
- Memory allocation with tainted size
- Pointer dereference with tainted offset
- Tainted division operand
- Tainted modulo operand
- Tainted NULL or non-null-terminated string
- Tainted sign change conversion
- Tainted size of variable length array
- Tainted string format
- Use of externally controlled environment variable
- Use of tainted pointer

```
#define SIZE 100
extern int tab[SIZE];

int taintedarrayindex(int num) {
    return tab[num];
}
```



□ Variable trace

```
30  return tab[num];
```

⊟ Result Review

Severity	High
Status	Fix

! ID 80: Array access with tainted index
Array index is from an unsecure source. Index

	Event
1	Formal parameter is a tainted value
2	! Array access with tainted index

Correction — Check Range Before Use
One possible correction is to check that `num` is in range before using it.

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE100 100
extern int tab[SIZE100];
static int tainted_int_source(void) {
        return strtol(getenv("INDEX"),NULL,10);
}
int taintedarrayindex(void) {
        int num = tainted_int_source();
        if (num >=
                re
        } else {
                re
        }
}
```

Result Information

Group: Tainted Data

Language: C | C++

Default: Off

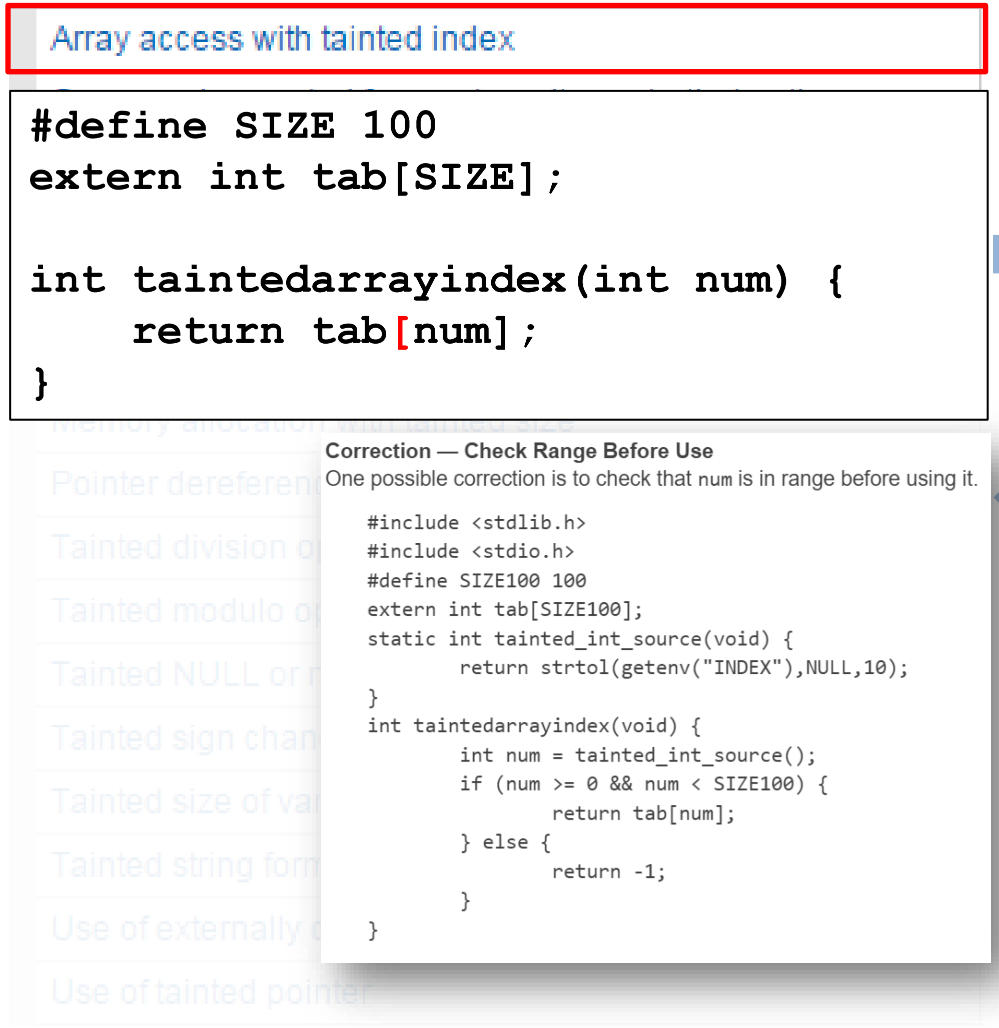
Command-Line Syntax: TAINTED_ARRAY_INDEX

Impact: Medium

CWE ID: 121, 124, 125, 129

Beyond Guidelines: Automated Taint Analysis

Defects related to data from an unsecure source



Array access with tainted index

```
#define SIZE 100
extern int tab[SIZE];

int taintedarrayindex(int num) {
    return tab[num];
}
```

Memory allocation with tainted size

Pointer dereferen

Tainted division o

Tainted modulo o

Tainted NULL or r

Tainted sign chan

Tainted size of va

Tainted string form

Use of externally c

Use of tainted pointer

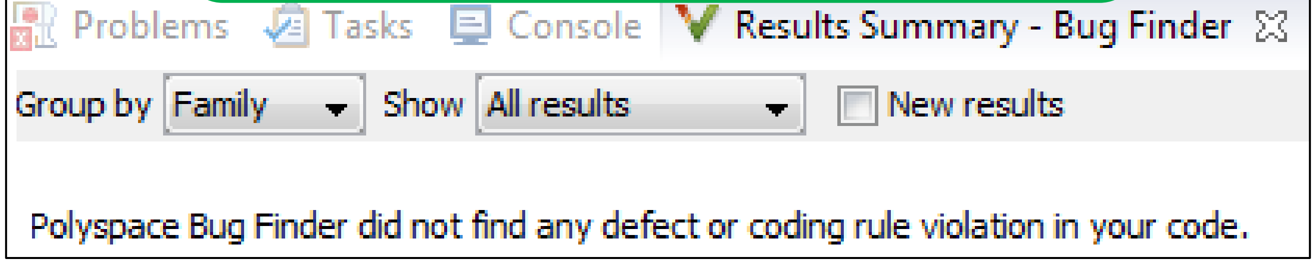
Correction — Check Range Before Use

One possible correction is to check that `num` is in range before using it.

```
#include <stdlib.h>
#include <stdio.h>
#define SIZE100 100
extern int tab[SIZE100];
static int tainted_int_source(void) {
        return strtol(getenv("INDEX"),NULL,10);
}
int taintedarrayindex(void) {
        int num = tainted_int_source();
        if (num >= 0 && num < SIZE100) {
                return tab[num];
        } else {
                return -1;
        }
}
```

```
#define SIZE 100
extern int tab[SIZE];

int taintedarrayindex(int num) {
    if (num >= 0 && num < SIZE) {
        return tab[num];
    } else {
        return -9999;
    }
}
```

Good to Go



Group by: Family

Show: All results

□ New results

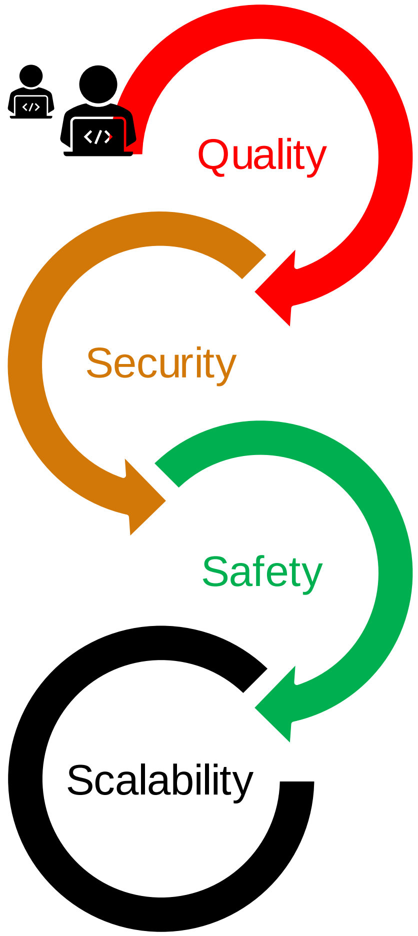
Polyspace Bug Finder did not find any defect or coding rule violation in your code.

Catching up with Cybersecurity in three steps:

1. Train developers…
   - Best practices & coding guidelines to avoid common errors
   - Distribute workload on the many, “shift left”

Miss “no” defects with static analysis…
   - *Sound* analysis is superior to Fuzz Testing
   - Considers all corner cases, guaranteed robustness

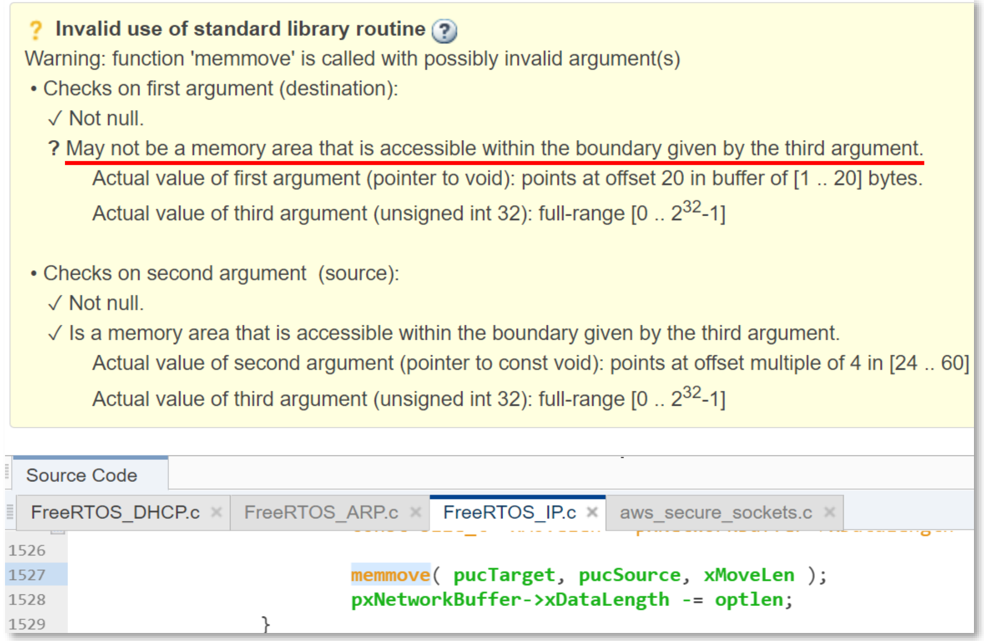
3. Automate, Collaborate & Monitor…
   - Rigorous “nightly security reviews” without experts
   - Central result storage & review
   - Quality gates to keep your software robust & clean



Why coding guidelines are good, but not enough

Many SAST tools only check “patterns”

Guideline passed != no vulnerabilities:



? **Invalid use of standard library routine** ?

Warning: function 'memmove' is called with possibly invalid argument(s)

- Checks on first argument (destination):
  - ✓ Not null.
  - ? May not be a memory area that is accessible within the boundary given by the third argument.
    - Actual value of first argument (pointer to void): points at offset 20 in buffer of [1 .. 20] bytes.
    - Actual value of third argument (unsigned int 32): full-range [0 .. 2^{32}-1]
- Checks on second argument (source):
  - ✓ Not null.
  - ✓ Is a memory area that is accessible within the boundary given by the third argument.
    - Actual value of second argument (pointer to const void): points at offset multiple of 4 in [24 .. 60]
    - Actual value of third argument (unsigned int 32): full-range [0 .. 2^{32}-1]

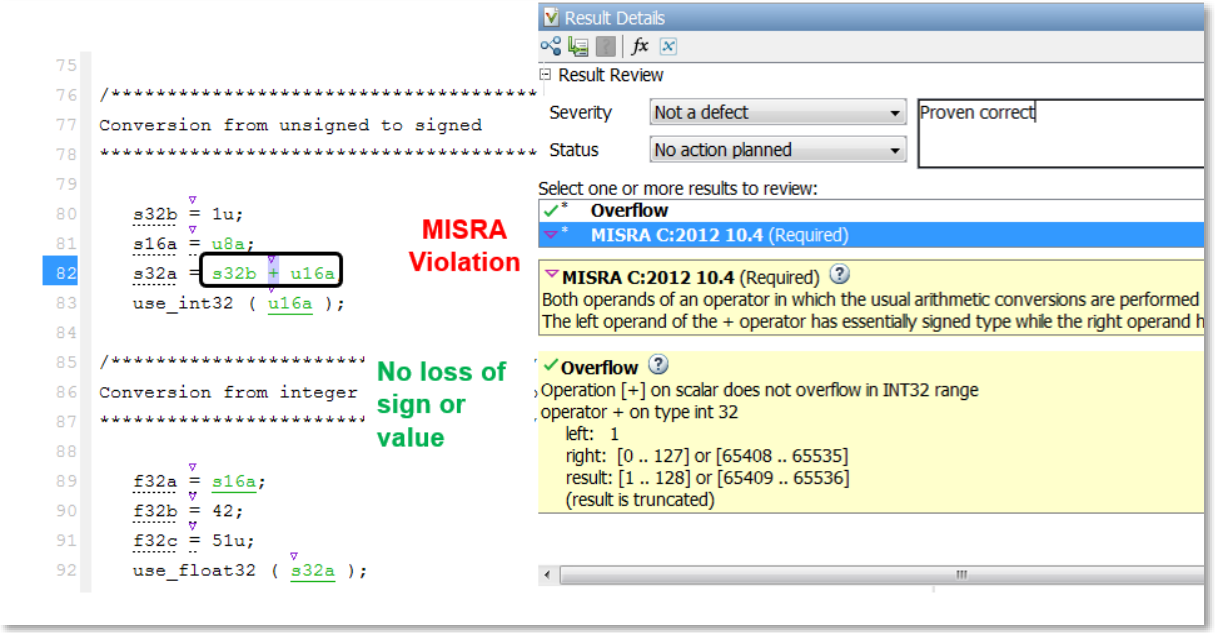
Source Code

FreeRTOS_DHCP.c × | FreeRTOS_ARP.c × | FreeRTOS_IP.c × | aws_secure_sockets.c ×

```
                        memmove( pucTarget, pucSource, xMoveLen );
                        pxNetworkBuffer->xDataLength -= optlen;
                    }
```

Inconsistent arguments to `memmove` → DoS!
Not checked by CERT/MISRA/…

Guideline violation != vulnerability:



```
/*****************************************
Conversion from unsigned to signed
*****************************************

    s32b = 1u;
    s16a = u8a;
    s32a = s32b + u16a;
    use_int32 ( u16a );

/*************************
Conversion from integer
*************************

    f32a = s16a;
    f32b = 42;
    f32c = 51u;
    use_float32 ( s32a );
```

MISRA Violation

No loss of sign or value

Result Details

Result Review

Severity: Not a defect

Status: No action planned

Proven correct

Select one or more results to review:

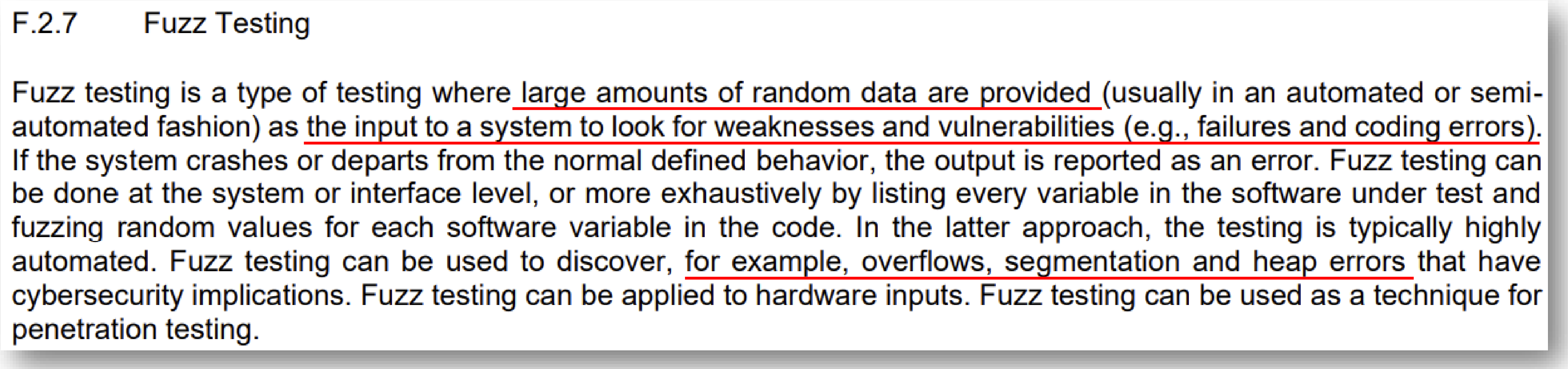
- ✓ * **Overflow**
- * **MISRA C:2012 10.4** (Required)

MISRA C:2012 10.4 (Required) ?
Both operands of an operator in which the usual arithmetic conversions are performed
The left operand of the + operator has essentially signed type while the right operand h

✓ **Overflow** ?
Operation [+] on scalar does not overflow in INT32 range
operator + on type int 32
left: 1
right: [0 .. 127] or [65408 .. 65535]
result: [1 .. 128] or [65409 .. 65536]
(result is truncated)

Valid mixing of different data types → No harm done!
Safe to ignore/justify MISRA violation.

Robustness “Testing” with Guarantees



F.2.7 Fuzz Testing

Fuzz testing is a type of testing where large amounts of random data are provided (usually in an automated or semi-automated fashion) as the input to a system to look for weaknesses and vulnerabilities (e.g., failures and coding errors). If the system crashes or departs from the normal defined behavior, the output is reported as an error. Fuzz testing can be done at the system or interface level, or more exhaustively by listing every variable in the software under test and fuzzing random values for each software variable in the code. In the latter approach, the testing is typically highly automated. Fuzz testing can be used to discover, for example, overflows, segmentation and heap errors that have cybersecurity implications. Fuzz testing can be applied to hardware inputs. Fuzz testing can be used as a technique for penetration testing.

From : ISO/SAE DIS 21434

- Through Fuzz testing
  - Requires execution on target → slow
  - Requires test harness → effort
  - E.g., (anti-)random testing, coverage testing, genetic algorithms

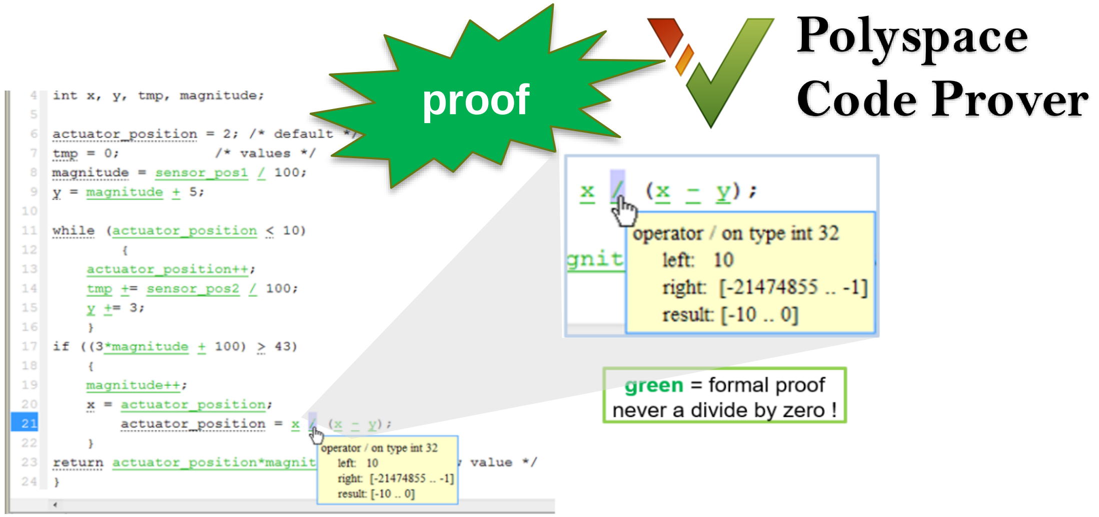
Not exhaustive → may miss vulnerabilities

Robustness “Testing” with Guarantees

- Through Fuzz testing
  - Requires execution on target → slow
  - Requires test harness → effort
  - E.g., (anti-)random testing, coverage testing, genetic algorithms

Not exhaustive → may miss vulnerabilities

- *Sound* static analysis with proof
  - Based on analysis, not execution
  - Requires no test harness
  - Considers all inputs & states
    - Boundary values, race conditions, sufficient checking of user inputs…?



```
int x, y, tmp, magnitude;

actuator_position = 2; /* default */
tmp = 0;              /* values */
magnitude = sensor_pos1 / 100;
y = magnitude + 5;

while (actuator_position < 10)
        {
    actuator_position++;
    tmp += sensor_pos2 / 100;
    y += 3;
    }
if ((3*magnitude + 100) > 43)
    {
    magnitude++;
    x = actuator_position;
        actuator_position = x / (x - y);
    }
return actuator_position*magnit         value */
}
```

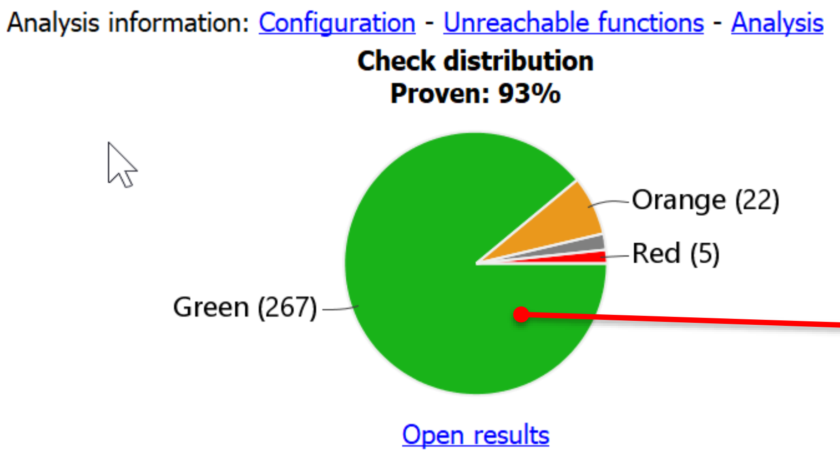
operator / on type int 32
left: 10
right: [-21474855 .. -1]
result: [-10 .. 0]

green = formal proof
never a divide by zero !

Miss no (checked) bugs → less vulnerabilities

Sound Static Application Security Testing (SAST) with Polyspace

Proof of robustness by analysis instead of evidence from dynamic execution

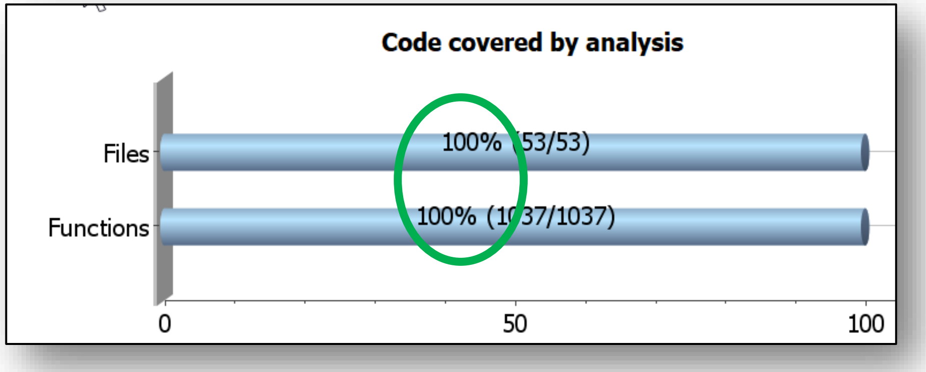


Analysis information: Configuration - Unreachable functions - Analysis

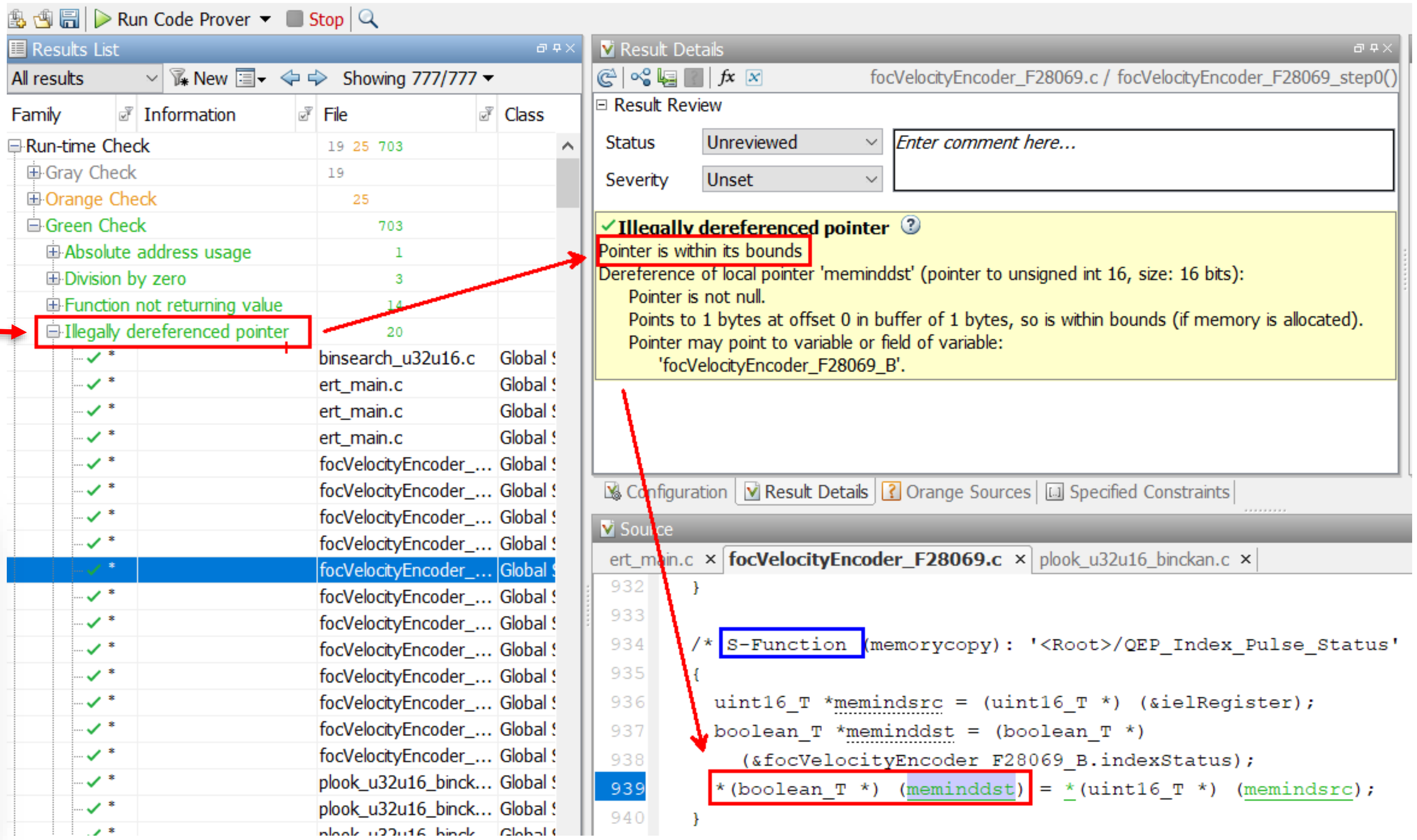
Check distribution

Proven: 93%

Open results



Code covered by analysis



Family	Information	File	Class
Run-time Check		19 25 703	
Gray Check		19	
Orange Check		25	
Green Check		703	
Absolute address usage		1	
Division by zero		3	
Function not returning value		14	
Illegally dereferenced pointer		20	
✓ *		binsearch_u32u16.c	Global !
✓ *		ert_main.c	Global !
✓ *		ert_main.c	Global !
✓ *		ert_main.c	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		focVelocityEncoder_...	Global !
✓ *		plook_u32u16_binck...	Global !
✓ *		plook_u32u16_binck...	Global !
✓ *		plook_u32u16_binck...	Global !

focVelocityEncoder_F28069.c / focVelocityEncoder_F28069_step0()

Result Review

Status: Unreviewed

Severity: Unset

Enter comment here...

✓ Illegally dereferenced pointer

Pointer is within its bounds

Dereference of local pointer 'meminddst' (pointer to unsigned int 16, size: 16 bits):

Pointer is not null.

Points to 1 bytes at offset 0 in buffer of 1 bytes, so is within bounds (if memory is allocated).

Pointer may point to variable or field of variable:

'focVelocityEncoder_F28069_B'.

```
}

/* S-Function (memorycopy): '<Root>/QEP_Index_Pulse_Status'
{
  uint16_T *memindsrc = (uint16_T *) (&ielRegister);
  boolean_T *meminddst = (boolean_T *)
    (&focVelocityEncoder_F28069_B.indexStatus);
  *(boolean_T *) (meminddst) = *(uint16_T *) (memindsrc);
}
```

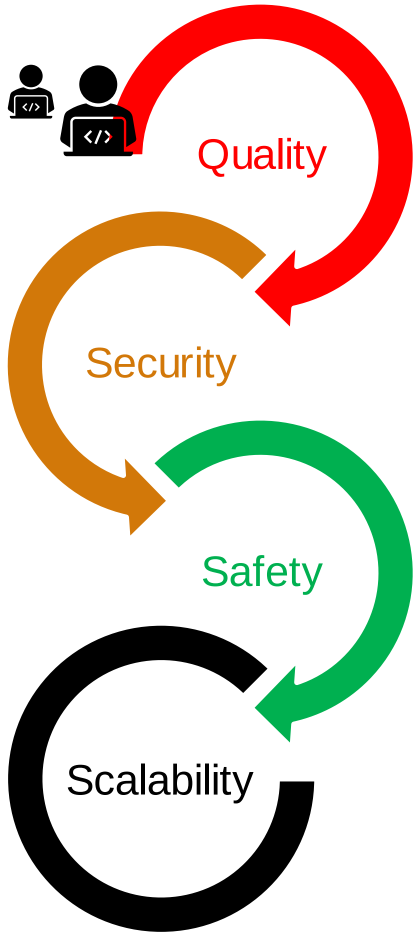
Considers *all* inputs & *all* program states, reduces need for Fuzz Testing

Catching up with Cybersecurity in three steps:

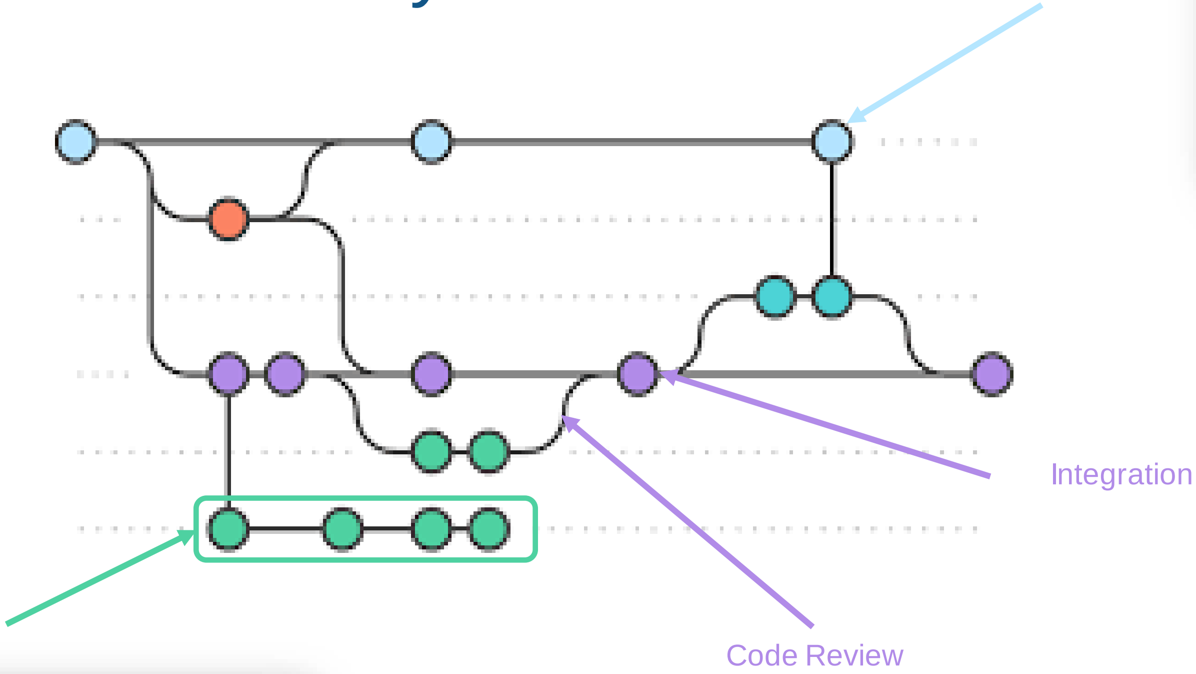
1. Train developers…
   - Best practices & coding guidelines to avoid common errors
   - Distribute workload on the many, “shift left”

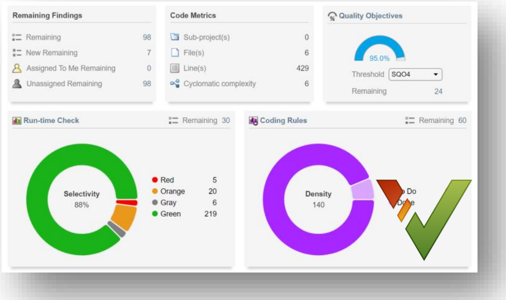
2. Miss “no” defects with static analysis…
   - *Sound* analysis is superior to Fuzz Testing
   - Considers all corner cases, guaranteed robustness

Automate, Collaborate & Monitor…
   - Rigorous “nightly security reviews” without experts
   - Central result storage & review
   - Quality gates to keep your software robust & clean



Continuous Vulnerability Verification





Remaining Findings	
Remaining	98
New Remaining	7
Assigned To Me Remaining	0
Unassigned Remaining	98

Code Metrics	
Sub-project(s)	0
File(s)	6
Line(s)	429
Cyclomatic complexity	6

Quality Objectives

95.0%

Threshold SQO4 ▾

Remaining 24

Run-time Check — Remaining 30

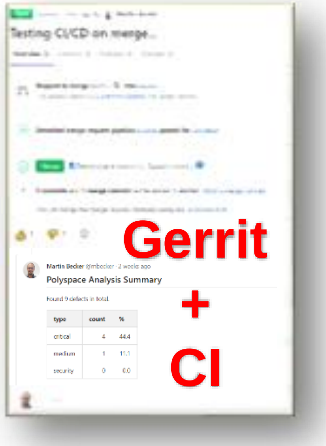
Selectivity 88%

● Red	5
● Orange	20
● Gray	6
● Green	219

Coding Rules — Remaining 60

Density 140

Reporting and Certification artifacts



Testing CI/CD on merge...

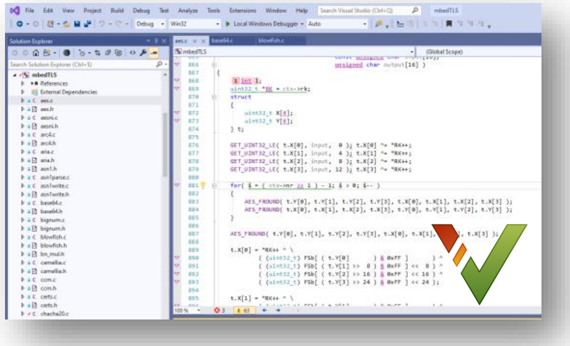
Martin Becker @mbecker · 2 weeks ago

Polyspace Analysis Summary

Found 9 defects in total.

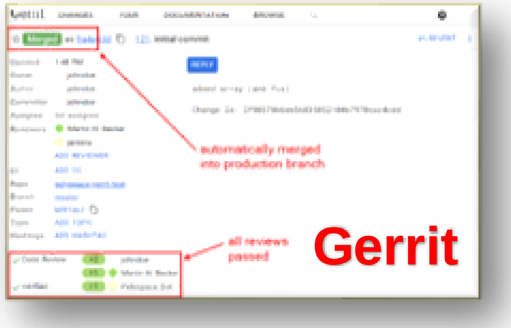
type	count	%
critical	4	44.4
medium	1	11.1
security	0	0.0

Automating quality gate into CI pipelines



```
                          unsigned char output[16] )
{
    int i;
    uint32_t *RK = ctx->rk;
    struct
    {
        uint32_t X[4];
        uint32_t Y[4];
    } t;

    GET_UINT32_LE( t.X[0], input,  0 ); t.X[0] ^= *RK++;
    GET_UINT32_LE( t.X[1], input,  4 ); t.X[1] ^= *RK++;
    GET_UINT32_LE( t.X[2], input,  8 ); t.X[2] ^= *RK++;
    GET_UINT32_LE( t.X[3], input, 12 ); t.X[3] ^= *RK++;

    for( i = ( ctx->nr >> 1 ) - 1; i > 0; i-- )
    {
        AES_FROUND( t.Y[0], t.Y[1], t.Y[2], t.Y[3], t.X[0], t.X[1], t.X[2], t.X[3] );
        AES_FROUND( t.X[0], t.X[1], t.X[2], t.X[3], t.Y[0], t.Y[1], t.Y[2], t.Y[3] );
    }

    AES_FROUND( t.Y[0], t.Y[1], t.Y[2], t.Y[3], t.X[0], t.X[1], t.X[2], t.X[3] );

    t.X[0] = *RK++ ^ \
            ( (uint32_t) FSb[ ( t.Y[0]       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( t.Y[1] >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( t.Y[2] >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( t.Y[3] >> 24 ) & 0xFF ] << 24 );

    t.X[1] = *RK++ ^ \
```

Direct developer feedback in IDE to fix security coding standards (CERT)

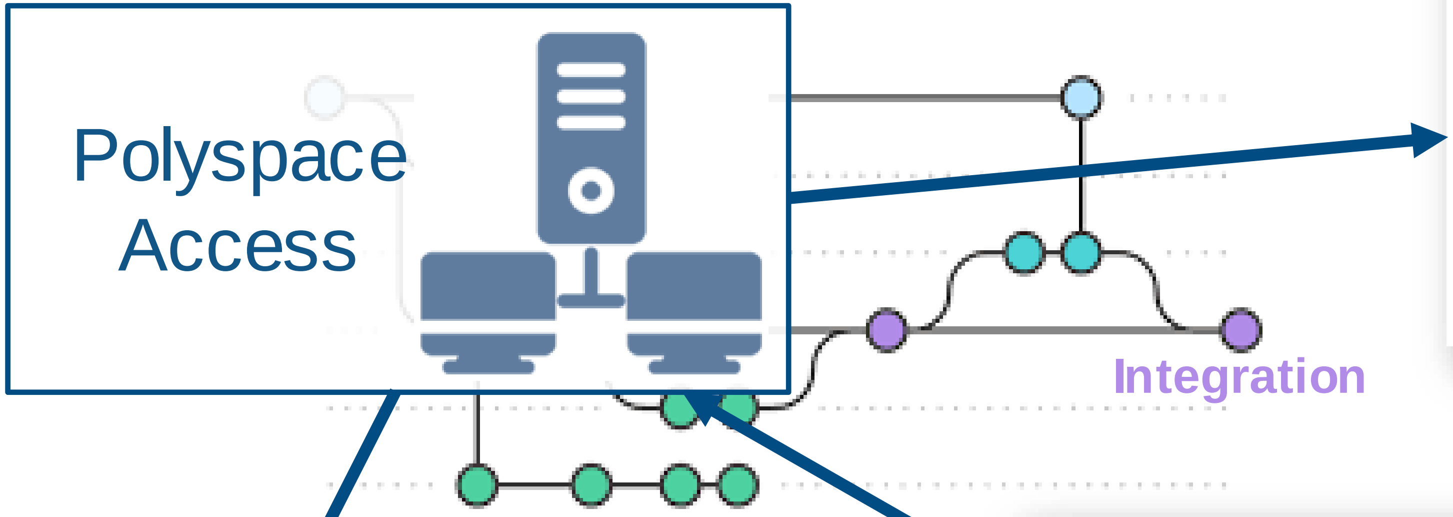


automatically merged into production branch

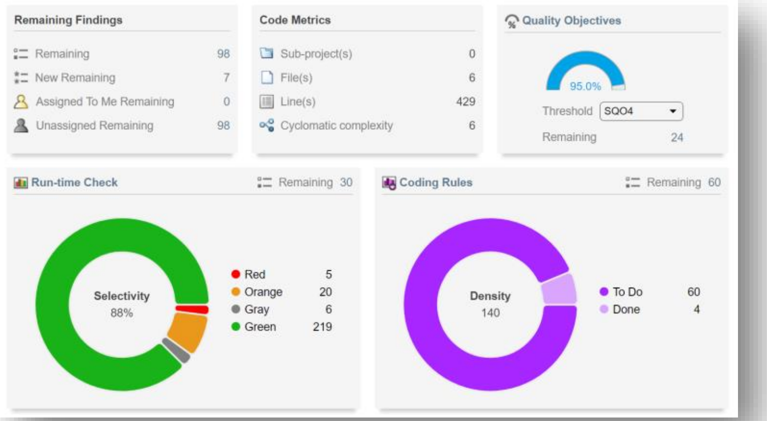
all reviews passed

Supporting unopinionated code reviews focusing on vulnerabilities

Cybersecurity Is Everyone Concern



Final VnV


Remaining Findings

Remaining	98
New Remaining	7
Assigned To Me Remaining	0
Unassigned Remaining	98

Code Metrics

Sub-project(s)	0
File(s)	6
Line(s)	429
Cyclomatic complexity	6

Quality Objectives

95.0%

Threshold: SQO4

Remaining 24

Run-time Check Remaining 30

Selectivity 88%

● Red	5
● Orange	20
● Gray	6
● Green	219

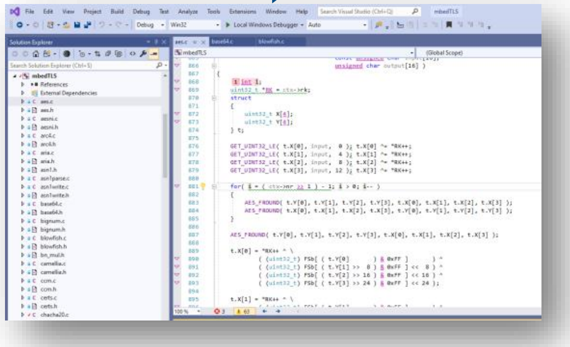
Coding Rules Remaining 60

Density 140

● To Do	60
● Done	4

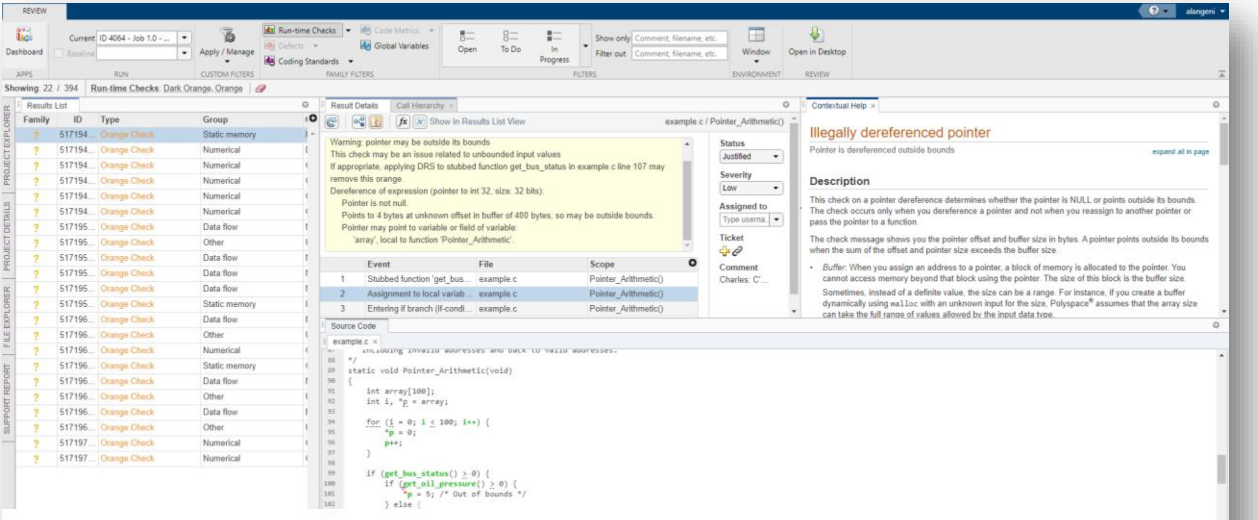
Define quality threshold
Populate reports with justification

Developer Branch



```
                                   unsigned char output[16] )
{
    int i;
    uint32_t *RK = ctx->rk;
    struct
    {
        uint32_t X[4];
        uint32_t Y[4];
    } t;

    GET_UINT32_LE( t.X[0], input,  0 ); t.X[0] ^= *RK++;
    GET_UINT32_LE( t.X[1], input,  4 ); t.X[1] ^= *RK++;
    GET_UINT32_LE( t.X[2], input,  8 ); t.X[2] ^= *RK++;
    GET_UINT32_LE( t.X[3], input, 12 ); t.X[3] ^= *RK++;

    for( i = ( ctx->nr >> 1 ) - 1; i > 0; i-- )
    {
        AES_FROUND( t.Y[0], t.Y[1], t.Y[2], t.Y[3], t.X[0], t.X[1], t.X[2], t.X[3] );
        AES_FROUND( t.X[0], t.X[1], t.X[2], t.X[3], t.Y[0], t.Y[1], t.Y[2], t.Y[3] );
    }

    AES_FROUND( t.Y[0], t.Y[1], t.Y[2], t.Y[3], t.X[0], t.X[1], t.X[2], t.X[3] );

    t.X[0] = *RK++ ^ \
            ( (uint32_t) FSb[ ( t.Y[0]       ) & 0xFF ]       ) ^
            ( (uint32_t) FSb[ ( t.Y[1] >>  8 ) & 0xFF ] <<  8 ) ^
            ( (uint32_t) FSb[ ( t.Y[2] >> 16 ) & 0xFF ] << 16 ) ^
            ( (uint32_t) FSb[ ( t.Y[3] >> 24 ) & 0xFF ] << 24 );

    t.X[1] = *RK++ ^ \
```

Focus on newly introduced vulnerabilities



Current: ID 4064 - Job 1.0 - ...

Show only: Comment, filename, etc.
Filter out: Comment, filename, etc.

Showing: 22 / 394 Run-time Checks: Dark Orange, Orange

Family	ID	Type	Group
?	517194...	Orange Check	Static memory
?	517194...	Orange Check	Numerical
?	517194...	Orange Check	Numerical
?	517194...	Orange Check	Numerical
?	517194...	Orange Check	Numerical
?	517194...	Orange Check	Numerical
?	517195...	Orange Check	Data flow
?	517195...	Orange Check	Other
?	517195...	Orange Check	Data flow
?	517195...	Orange Check	Data flow
?	517195...	Orange Check	Data flow
?	517195...	Orange Check	Static memory
?	517196...	Orange Check	Data flow
?	517196...	Orange Check	Other
?	517196...	Orange Check	Numerical
?	517196...	Orange Check	Static memory
?	517196...	Orange Check	Data flow
?	517196...	Orange Check	Other
?	517196...	Orange Check	Data flow
?	517196...	Orange Check	Other
?	517197...	Orange Check	Numerical
?	517197...	Orange Check	Numerical

example.c / Pointer_Arithmetic()

Warning: pointer may be outside its bounds
This check may be an issue related to unbounded input values
If appropriate, applying DRS to stubbed function get_bus_status in example.c line 107 may remove this orange.
Dereference of expression (pointer to int 32, size: 32 bits):
Pointer is not null.
Points to 4 bytes at unknown offset in buffer of 400 bytes, so may be outside bounds.
Pointer may point to variable or field of variable:
'array', local to function 'Pointer_Arithmetic'.

	Event	File	Scope
1	Stubbed function 'get_bus...	example.c	Pointer_Arithmetic()
2	Assignment to local variab...	example.c	Pointer_Arithmetic()
3	Entering if branch (if-condi...	example.c	Pointer_Arithmetic()

Status: Justified
Severity: Low
Assigned to: Type userna...
Ticket
Comment: Charles: C'...

Illegally dereferenced pointer

Pointer is dereferenced outside bounds

Description

This check on a pointer dereference determines whether the pointer is NULL or points outside its bounds. The check occurs only when you dereference a pointer and not when you reassign to another pointer or pass the pointer to a function.

The check message shows you the pointer offset and buffer size in bytes. A pointer points outside its bounds when the sum of the offset and pointer size exceeds the buffer size.

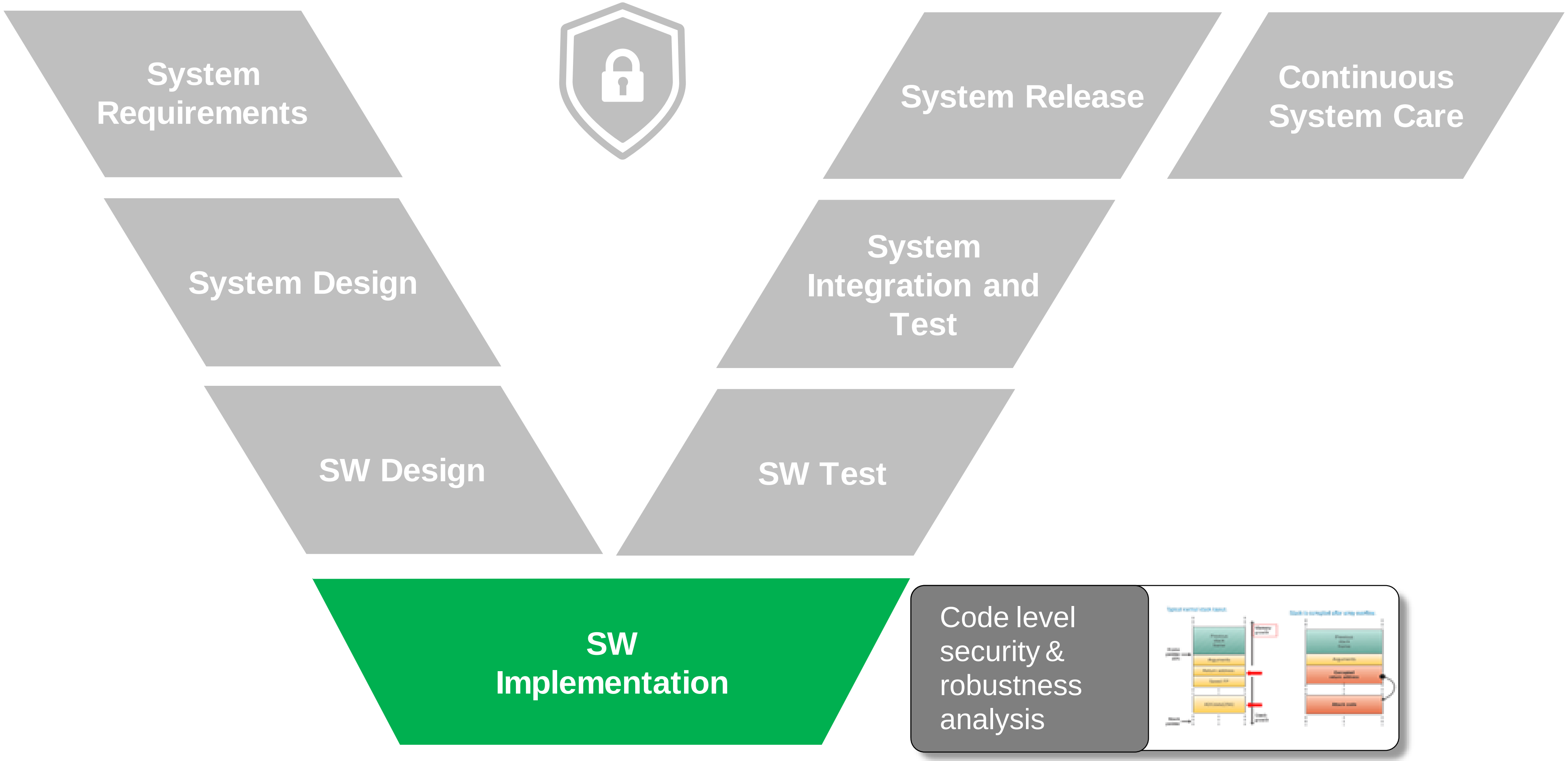
- *Buffer*: When you assign an address to a pointer, a block of memory is allocated to the pointer. You cannot access memory beyond that block using the pointer. The size of this block is the buffer size.

  Sometimes, instead of a definite value, the size can be a range. For instance, if you create a buffer dynamically using `malloc` with an unknown input for the size, Polyspace® assumes that the array size can take the full range of values allowed by the input data type.

```
   including invalid addresses and back to valid addresses.
*/
static void Pointer_Arithmetic(void)
{
    int array[100];
    int i, *p = array;

    for (i = 0; i < 100; i++) {
        *p = 0;
        p++;
    }

    if (get_bus_status() > 0) {
        if (get_oil_pressure() > 0) {
            *p = 5; /* Out of bounds */
        } else {
```

Collaborative vulnerabilities review
Integrate with Issue tracking tool(Jira, Redmine)

MathWorks Capabilities for Cybersecurity



System Requirements

System Design

SW Design

SW Implementation

SW Test

System Integration and Test

System Release

Continuous System Care

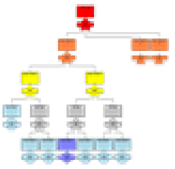
Code level security & robustness analysis

MathWorks Capabilities for Cybersecurity

System Requirements

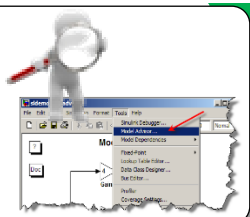
System Design

Allocate sec reqs, threat/risk analysis



SW Design

Secure design & modeling

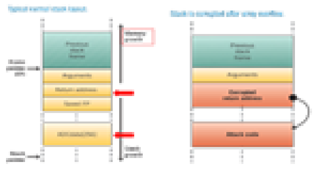


SW Implementation

Secure code generation & deployment

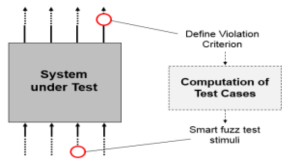
- MISRA C/C++
- CERT C/C++
- CWE
- TS 17961

Code level security & robustness analysis



SW Test

Req. testing, Fuzz testing, Attack Sim.



System Integration and Test

Int. Tests & intrusion detection

Intrusion

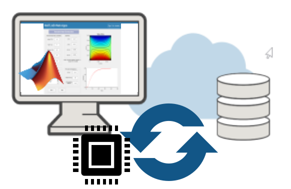
Detection

Reaction

System Release

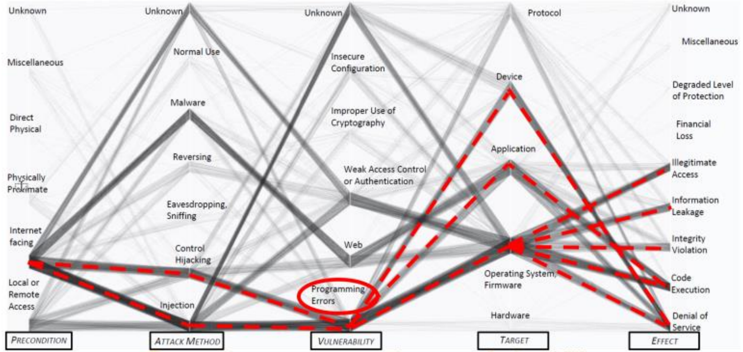
Continuous System Care

SOC Data Analytics, Design Updates

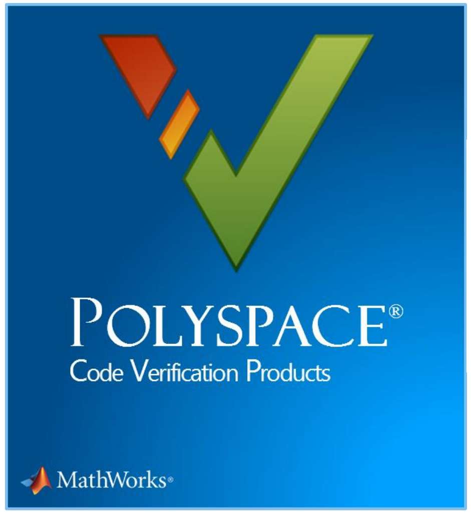


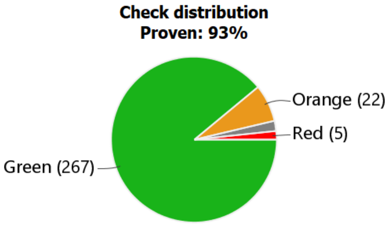
Key Takeaways

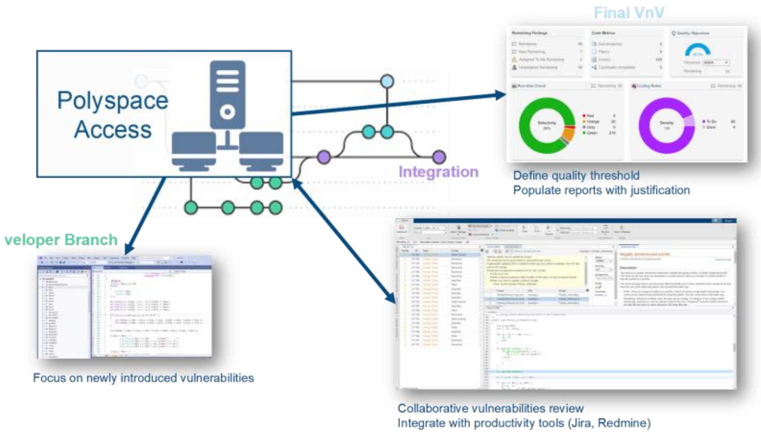
- Achieve Higher Security Level with Polyspace Products
- Prove Absence of Critical Vulnerabilities to Reduce Testing Effort
- Raise Team Skills to Tackle Vulnerabilities



Programming errors are one major source of vulnerabilities







Define quality threshold
Populate reports with justification

Focus on newly introduced vulnerabilities

Collaborative vulnerabilities review
Integrate with productivity tools (Jira, Redmine)

MATLAB EXPO
2021

Thank you

MathWorks®