

MATLAB EXPO 2017

KOREA

4월 27일, 서울

등록 하기 matlabexpo.co.kr

Polyspace를 활용한 MISRA C:2012 및 실행시간 오류 검사

Introduction to Polyspace with MISRA C:2012 and RTE

유용출 과장

Gary.Ryu@mathworks.co.kr

Agenda

- Why do we check MISRA C and Runtime errors?
- Polyspace Introduction
 - How to check MISRA C:2012 violations
 - How to verify Runtime errors

Why do we check MISRA C or Runtime error ?

The intention was to provide a **"restricted subset of a standardized structured language"** as required in the 1994 MISRA Guidelines for automotive systems being developed to **meet the requirements of functional safety standards like ISO 26262.**

Table 1 – Topics To Be Covered By Modeling and Coding Guidelines

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++	++	++	++
1b	Use of language subsets	++	++	++	++
1c	Enforcement of strong typing	++	++	++	++
1d	Use of defensive implementation techniques	0	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	
1g	Use of style guides	+	++	++	
1h	Use of naming conventions	++	++	++	

Table 9 – Methods for Verification of Software Unit Design and Implementation

Topics		ASIL			
		A	B	C	D
1a	Walkthrough	++	+	0	0
1b	Inspection	+	++	++	++
1c	Semiformal verification	+	+	++	++
1d	Formal verification	0	0	+	+
1e	Control flow analysis	+	+	++	++
1f	Data flow analysis	+	+	++	++
1g	Static code analysis	+	++	++	++
1h	Semantic code analysis	+	+	+	+

Why **restricted subset**?

- There are several drawbacks with the C language
 - ISO Standard language definition is *incomplete* ...
 - Undefined behavior
 - Unspecified behavior
 - Implementation-defined behavior
 - Misuse language
 - Misunderstanding language
 - Lack of Runtime error checking
- One of solution is **MISRA C** and **RTE detection** with **Static Analysis**

Why restricted subset?

```
int foo (int arg) {  
    return arg + 1;  
}  
  
void main (void) {  
    int var = 0;  
    printf ("var : %d and %d\n", var++, foo(var));  
}
```

Output with ...

- gcc 5.4.0 : var : 0 and 1
- Visual Studio 2013 : var : 0 and 2

Why **restricted subset**?

- There are several drawbacks with the C language
 - ISO Standard language definition is *incomplete* ...
 - Undefined behavior
 - Unspecified behavior
 - Implementation-defined behavior
 - Misuse language
 - Misunderstanding language
 - Lack of Runtime error checking
- One of solution is **MISRA C** and **RTE detection** with **Static Analysis**

Brief History of MISRA C

- **MISRA C:2012**

- Compatible with **ISO/IEC 9899:1999 (C99)**
- published in 2013
- 159 Guidelines
 - 16 Directives
 - 143 Rules



- 173 Guidelines
 - 17 Directives
 - 156 Rules



- **MISRA C:2004**

- Compatible with ISO/IEC 9899:1990 (C90)

- **MISRA C:1998**

- Compatible with ISO/IEC 9899:1990 (C90)

What is MISRA C:2012

- Directives

Guidelines for which it is not possible to provide the full description necessary to perform a check for compliance. Static analysis tools may be able to assist in checking compliance. For example, items are checked with design documents or requirements specification.

- Rules

Guidelines for which a complete description has been provided. It is possible to check compliance with source code without any other information.

What is MISRA C:2012

■ Directives

- 17 Directives
 - 10 Required directives
 - 7 Advisory directives

■ Rules

- 156 Rules
 - 16 Mandatory rules
 - 108 Required rules
 - 32 Advisory rules

Mandatory:

- Deviation from this guidelines is not permitted.

Required:

- Formal deviation is required.

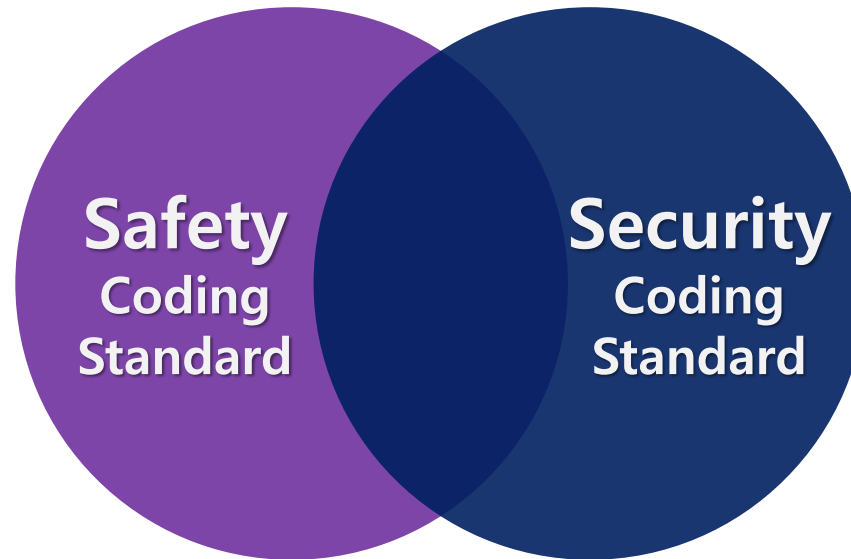
Advisory:

- Formal deviation is not necessary, but alternative arrangements should be made.

* Any guideline can be treated as required/mandatory guideline.

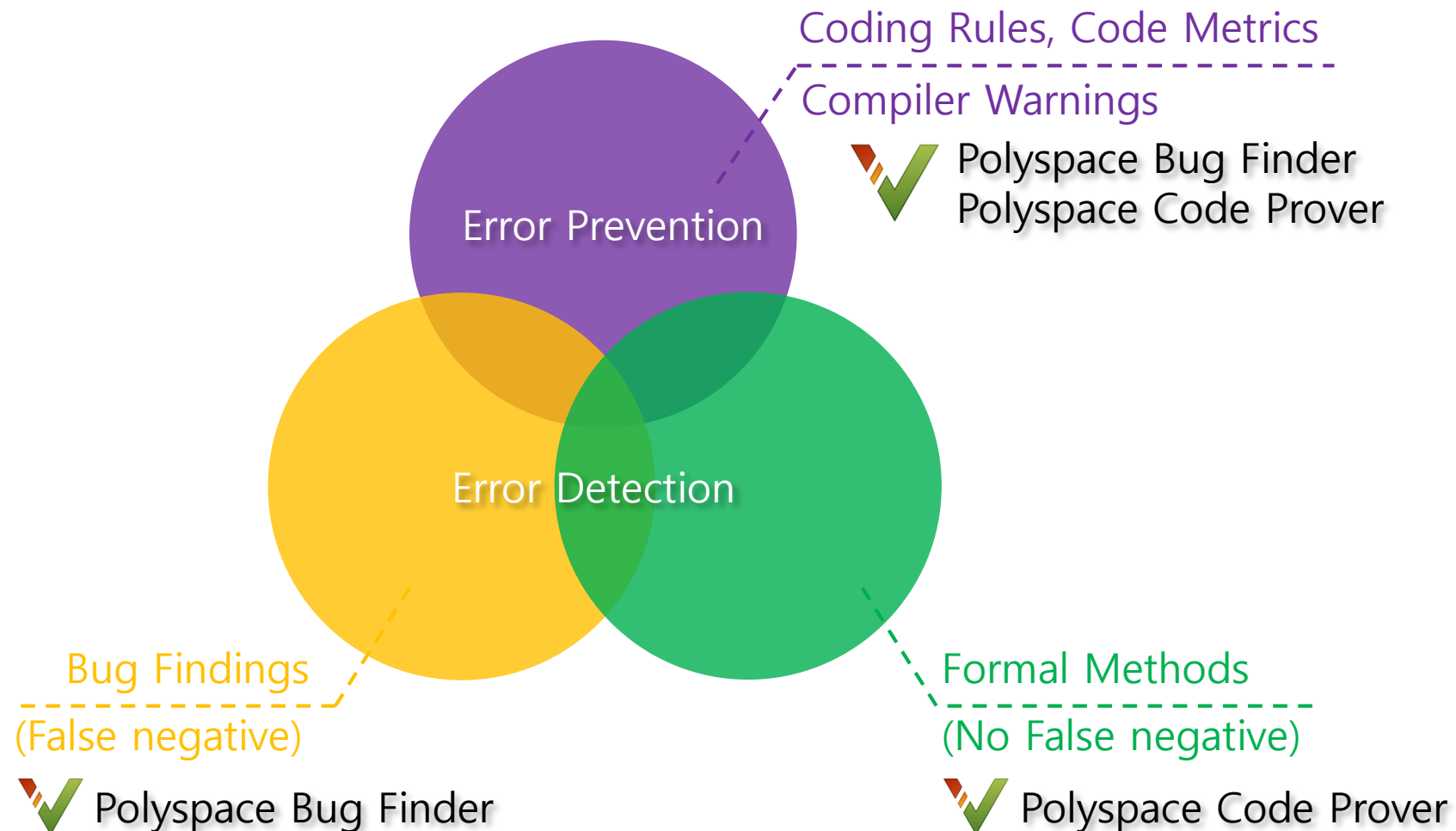
New Security guidelines of MISRA C:2012

- are to improve the coverage of the **security concerns** highlighted by **ISO/IEC 17961:2013**

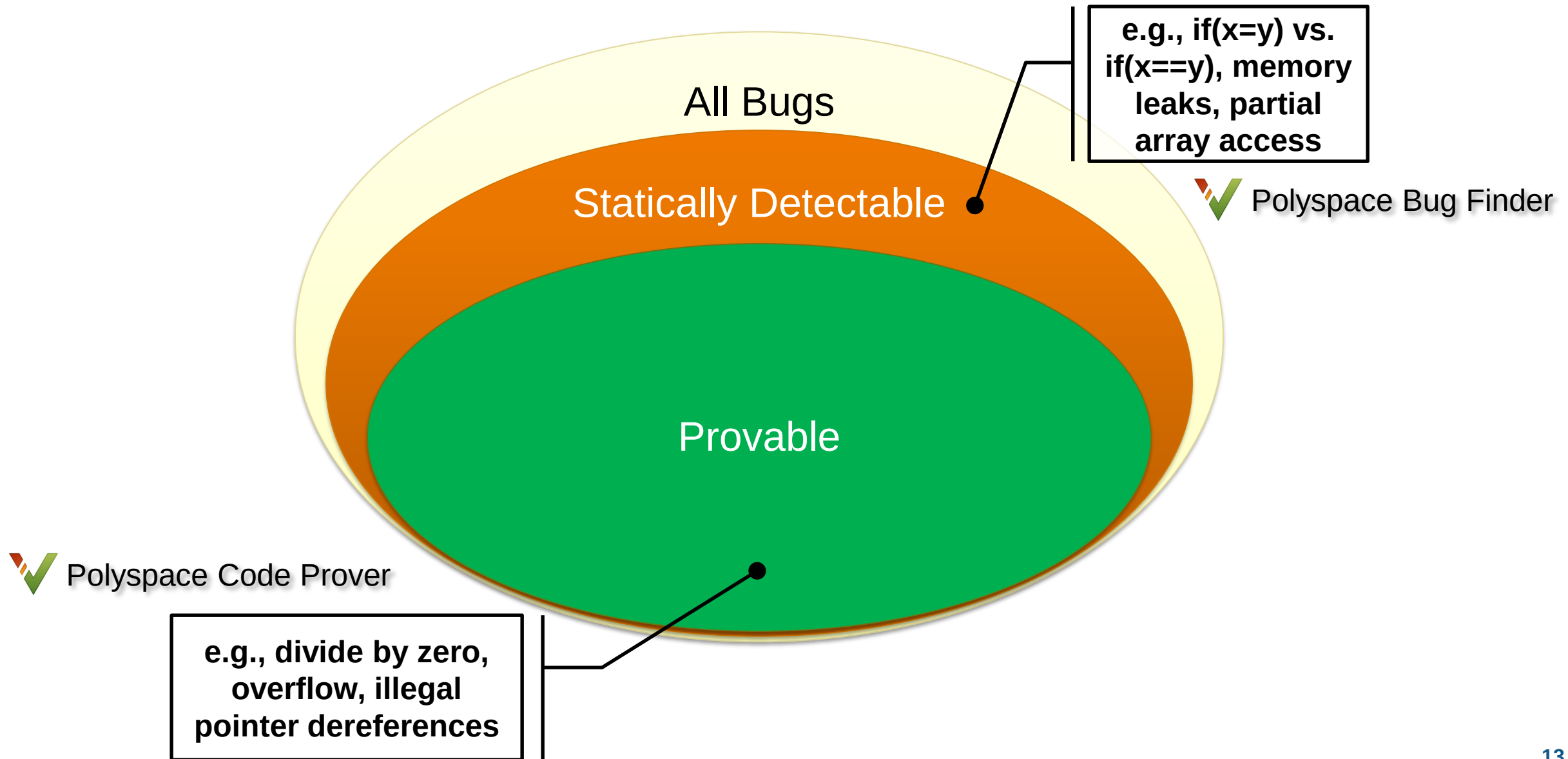


- MISRA C has evolved...
from automotive standard to industry-wide standard!

Polyspace PRODUCTS



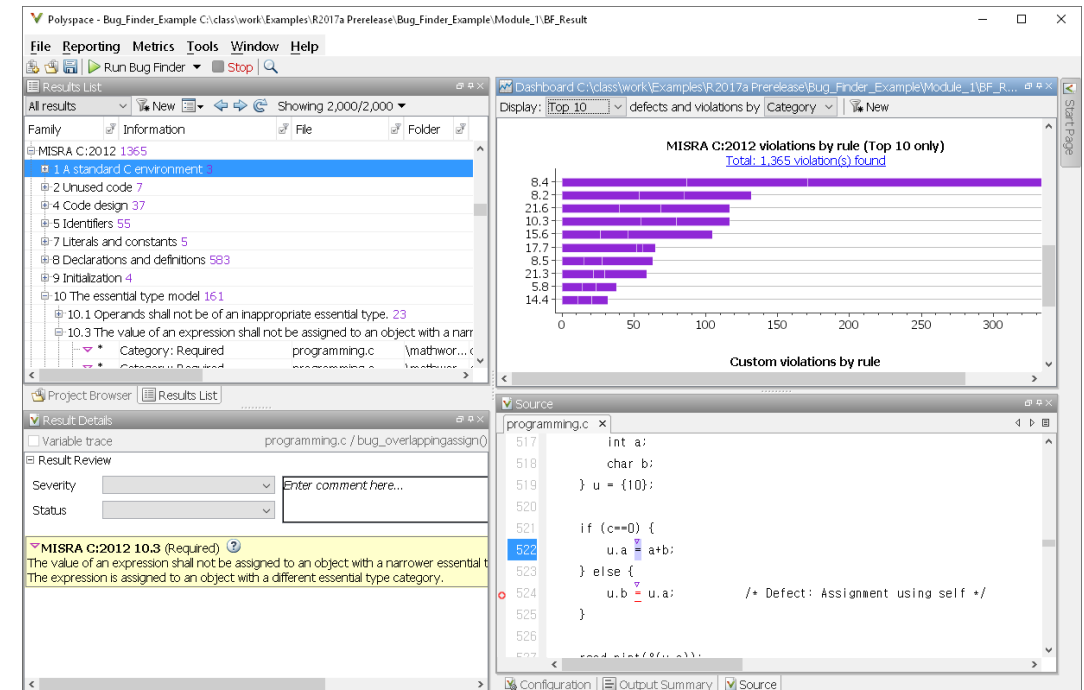
Not all bugs can be statically proven



Polyspace supports for Coding Rules Compliance



- **MISRA C:2012**
 - 11 Directives supported
 - 156 rules supported
 - 6 directives not enforceable
- **MISRA C++:2008**
 - 185 of the 228 rules supported
- **JSF++:2005**
 - 157 of 234 rules supported



Polyspace supports for various Code Metrics

Project Metrics

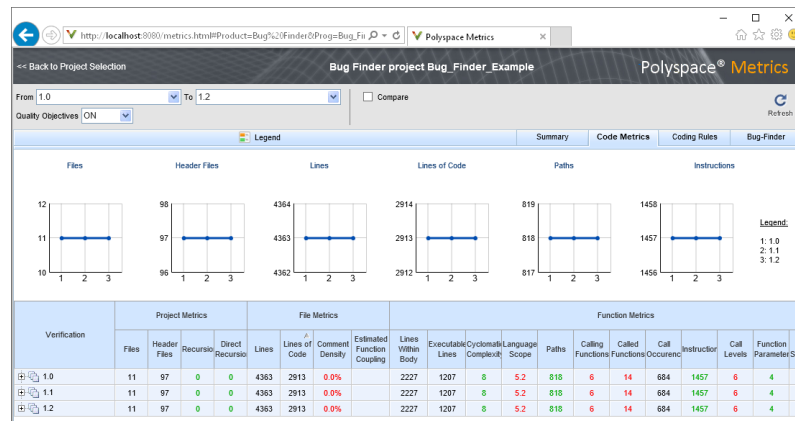
- Direct Recursions
- Header Files
- Files
- Recursions

File Metrics

- Comment Density
- Estimated Function Coupling
- Lines
- Lines without comment

Function Metrics

- Cyclomatic Complexity
- Higher Estimate of Local Variable Size
- Lower Estimate of Local Variable Size
- Language Scope
- Call Levels
- Call Occurrences
- Called Functions
- Calling Functions
- Executable Lines
- Function Parameters
- Goto Statements
- Instructions
- Lines Within Body
- Local Non-Static Variables
- Local Static Variables
- Paths
- Return Statements



Types of Defects detected by Polyspace Bug Finder

Numerical

- Division by zero, Overflow
- Invalid use of standard library integer/floating point routine
- ...

Static memory

- Array access out of bounds
- Null pointer
- ...

Dynamic memory

- Memory leaks
- Use of previously freed pointer
- ...

Dataflow

- Write without further read
- Non-initialized variable
- ...

Concurrency

- Data races (atomic, non-atomic)
- Deadlocks
- ...

Resource management

- Resource leak
- Writing to read-only resource
- ...

Programming

- Invalid use of = or == operator
- Declaration mismatch
- ...

Good Practice

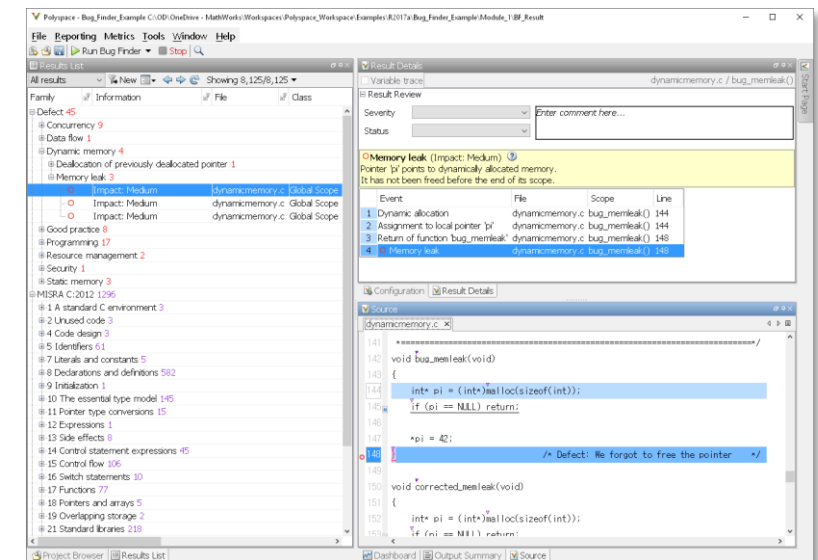
- Unused parameter
- Large pass-by-value argument
- ...

Security

- Unsafe standard function
- Use of non-secure temporary file
- ...

Tainted data

- Array access with tainted index
- Tainted sign change conversion
- ...



www.mathworks.com/help/bugfinder/defect-reference.html

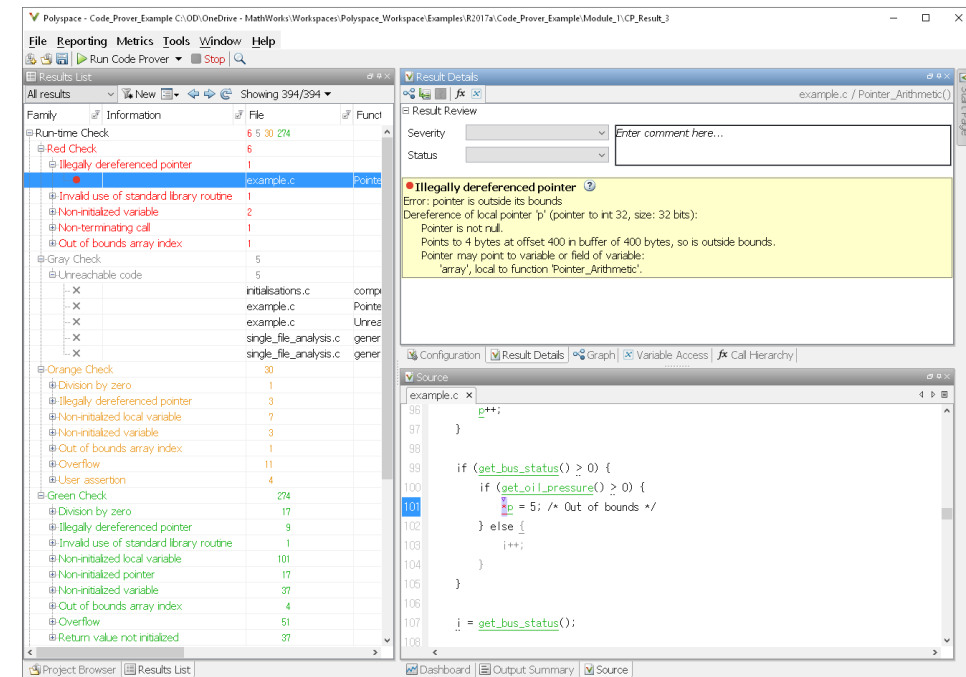
Full list of Runtime checks in Polyspace Code Prover

C run-time checks

- Unreachable Code
- Function not called
- Function not reachable
- Non-initialized local variable
- Non-initialized pointer
- Non-initialized variable
- Return value not initialized
- Division by zero
- Invalid operation on floats
- Invalid shift operations
- Overflow
- Subnormal float
- Absolute address usage
- Illegally dereferenced pointer
- Out of bound array index
- Non-terminating call
- Non-terminating loop
- Correctness condition (array conversion must not extend range, function pointer does not point to a valid function)
- Invalid use of standard library routine
- User assertion

Additional run-time checks for C++ only

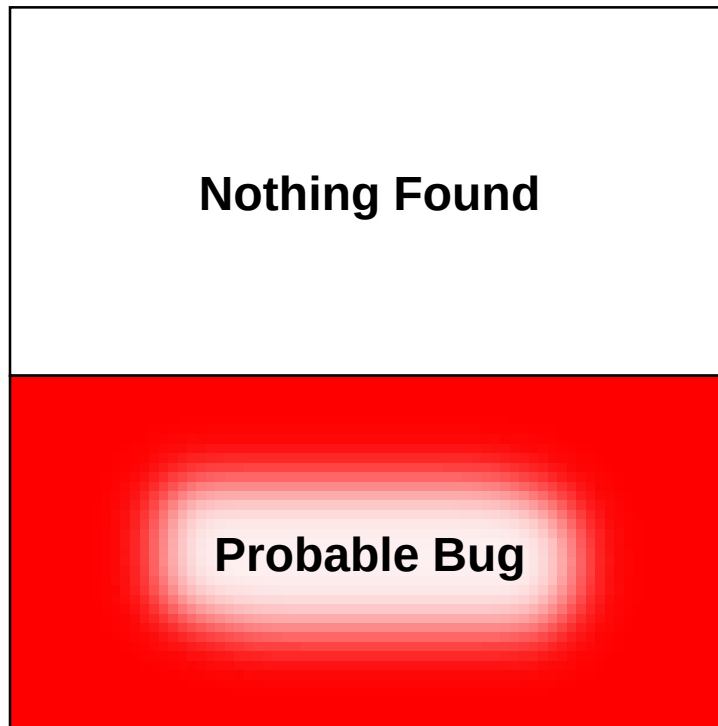
- Incorrect object oriented programming
- Invalid C++ specific operations
- Function not returning value
- Null this-pointer calling method
- Uncaught exception



www.mathworks.com/help/codeprover/run-time-check-reference.html

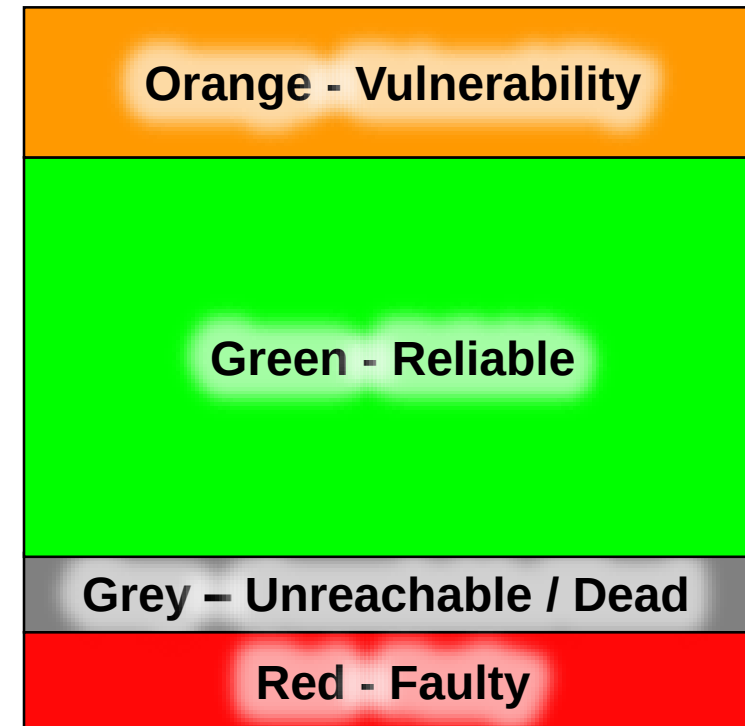
How do Bug Finder results differ from Code Prover results?

Bug Finder



 S.

Code Prover



Purple - coding rule violations

Polyspace demonstration

