EXP32-C. Do not access a volatile object through a nonvolatile reference

An object that has volatile-qualified type may be modified in ways unknown to the implementation or have other unknown side effects. Referencing a volatile object by using a non-volatile lvalue is undefined behavior. The C Standard, 6.7.3 [ISO/IEC 9899:2011], states

> *If an attempt is made to refer to an object defined with a volatile-qualified type through use of an lvalue with non-volatile-qualified type, the behavior is undefined.*

See undefined behavior 65.

Noncompliant Code Example

In this noncompliant code example, a volatile object is accessed through a non-volatile-qualified reference, resulting in undefined behavior:

```
#include <stdio.h>

void func(void) {
  static volatile int **ipp;
  static int *ip;
  static volatile int i = 0;

  printf("i = %d.\n", i);

  ipp = &ip; /* May produce a warning diagnostic */
  ipp = (int**) &ip; /* Constraint violation; may produce a warning diagnostic */
  *ipp = &i; /* Valid */
  if (*ip != 0) { /* Valid */
    /* ... */
  }
}
```

The assignment `ipp = &ip` is not safe because it allows the valid code that follows to reference the value of the volatile object `i` through the non-volatile-qualified reference `ip`. In this example, the compiler may optimize out the entire `if` block because `*ip != 0` must be false if the object to which `ip` points is not volatile.

Implementation Details

This example compiles without warning on Microsoft Visual Studio 2013 when compiled in C mode (`/TC`) but causes errors when compiled in C++ mode (`/TP`).

GCC 4.8.1 generates a warning but compiles successfully.

Compliant Solution

In this compliant solution, `ip` is declared `volatile`:

```
#include <stdio.h>

void func(void) {
  static volatile int **ipp;
  static volatile int *ip;
  static volatile int i = 0;

  printf("i = %d.\n", i);

  ipp = &ip;
  *ipp = &i;
  if (*ip != 0) {
    /* ... */
  }

}
```

Risk Assessment

Accessing an object with a volatile-qualified type through a reference with a non-volatile-qualified type is undefined behavior.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
EXP32-C	Low	Likely	Medium	P6	L2

Automated Detection

Tool	Version	Checker	Description
Astrée	19.04	**pointer-qualifier-cast-volatile** **pointer-qualifier-cast-volatile-implicit**	Supported indirectly via MISRA C 2012 Rule 11.8
Axivion Bauhaus Suite	6.9.0	**CertC-EXP32**	Fully implemented
Clang	3.9	`-Wincompatible-pointer-types-discards-qualifiers`	
Compass/ROSE			
Coverity	2017.07	**MISRA C 2012 Rule 11.8**	Implemented
GCC	4.3.5		Can detect violations of this rule when the `-Wcast-qual` flag is used
LDRA tool suite	9.7.1	**344 S**	Partially implemented
Parasoft C/C++test	10.4.2	**CERT_C-EXP32-a**	A cast shall not remove any 'const' or 'volatile' qualification from the type of a pointer or reference
Polyspace Bug Finder	R2019b	CERT C: Rule EXP32-C	Checks for cast to pointer that removes const or volatile qualification (rule fully covered)
PRQA QA-C	9.7	**0312,562,563,673,674**	Fully implemented
RuleChecker	19.04	**pointer-qualifier-cast-volatile** **pointer-qualifier-cast-volatile-implicit**	Supported indirectly via MISRA C 2012 Rule 11.8

Related Vulnerabilities

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

Related Guidelines

Key here (explains table format and definitions)

Taxonomy	Taxonomy item	Relationship
ISO/IEC TR 24772:2013	Pointer Casting and Pointer Type Changes [HFC]	Prior to 2018-01-12: CERT: Unspecified Relationship
ISO/IEC TR 24772:2013	Type System [IHN]	Prior to 2018-01-12: CERT: Unspecified Relationship
MISRA C:2012	Rule 11.8 (required)	Prior to 2018-01-12: CERT: Unspecified Relationship
CERT C	EXP55-CPP. Do not access a cv-qualified object through a cv-unqualified type	Prior to 2018-01-12: CERT: Unspecified Relationship

Bibliography

[ISO/IEC 9899:2011]	6.7.3, "Type Qualifiers"