MEM10-C. Define and use a pointer validation function

Many functions accept pointers as arguments. If the function dereferences an invalid pointer (as in EXP34-C. Do not dereference null pointers) or reads or writes to a pointer that does not refer to an object, the results are undefined. Typically, the program will terminate abnormally when an invalid pointer is dereferenced, but it is possible for an invalid pointer to be dereferenced and its memory changed without abnormal termination [Jack 2007]. Such programs can be difficult to debug because of the difficulty in determining if a pointer is valid.

One way to eliminate invalid pointers is to define a function that accepts a pointer argument and indicates whether or not the pointer is valid for some definition of valid. For example, the following function declares any pointer to be valid except NULL:

```c
#include <stdio.h>
#include <stdlib.h>

int valid(void *ptr) {
    return (ptr != NULL);
}
```

Some platforms have platform-specific pointer validation tools.

The following code relies on the _etext address, defined by the loader as the first address following the program text on many platforms, including AIX, Linux, QNX, IRIX, and Solaris. It is not POSIX-compliant, nor is it available on Windows.

```c
#include <stdio.h>
#include <stdlib.h>

int valid(void *p) {
    extern char _etext;
    return (p != NULL) && ((char*) p > &_etext);
}

int main(void) {
    int local;
    printf("pointer to local var valid? %d\n", valid(&local));
    printf("pointer to static var valid? %d\n", valid(&global));
    printf("pointer to function valid? %d\n", valid((void *)main));
    int *p = (int *) malloc(sizeof(int));
    printf("pointer to heap valid? %d\n", valid(p));
    printf("pointer to end of allocated heap valid? %d\n", valid(++p));
    free(--p);
    printf("pointer to freed heap valid? %d\n", valid(p));
    printf("null pointer valid? %d\n", valid(NULL));
    return 0;
}
```

On a Linux platform, this program produces the following output:

```
pointer to local var valid? 1
pointer to static var valid? 1
pointer to function valid? 0
pointer to heap valid? 1
pointer to end of allocated heap valid? 1
pointer to freed heap valid? 1
null pointer valid? 0
```

The valid() function does not guarantee validity; it only identifies null pointers and pointers to functions as invalid. However, it can be used to catch a substantial number of problems that might otherwise go undetected.

Noncompliant Code Example

In this noncompliant code example, the incr() function increments the value referenced by its argument. It also ensures that its argument is not a null pointer. But the pointer could still be invalid, causing the function to corrupt memory or terminate abnormally.
void incr(int *intptr) {
    if (intptr == NULL) {
        /* Handle error */
    }
    (*intptr)++;
}

Compliant Solution

This incr() function can be improved by using the valid() function. The resulting implementation is less likely to dereference an invalid pointer or write to memory that is outside the bounds of a valid object.

```c
void incr(int *intptr) {
    if (!valid(intptr)) {
        /* Handle error */
    }
    (*intptr)++;
}
```

The valid() function can be implementation dependent and perform additional, platform-dependent checks when possible. In the worst case, the valid() function may only perform the same null-pointer check as the noncompliant code example. However, on platforms where additional pointer validation is possible, the use of a valid() function can provide checks.

Risk Assessment

A pointer validation function can be used to detect and prevent operations from being performed on some invalid pointers.

<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>MEM10-C</td>
<td>High</td>
<td>Unlikely</td>
<td>High</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>LDRA tool suite</td>
<td>9.7.1</td>
<td>159 S</td>
<td>Enhanced enforcement</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Related Guidelines

<table>
<thead>
<tr>
<th>SEI CERT C++ Coding Standard</th>
<th>VOID MEM10-CPP. Define and use a pointer validation function</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITRE CWE</td>
<td>CWE-20, Improper Input Validation</td>
</tr>
<tr>
<td></td>
<td>CWE-79, Improper Neutralization of Input During Web Page Generation (‘Cross-site Scripting’)</td>
</tr>
<tr>
<td></td>
<td>CWE-89, Improper Neutralization of Special Elements used in an SQL Command (‘SQL Injection’)</td>
</tr>
<tr>
<td></td>
<td>CWE-91, XML Injection (aka Blind XPath Injection)</td>
</tr>
<tr>
<td></td>
<td>CWE-94, Improper Control of Generation of Code (‘Code Injection’)</td>
</tr>
<tr>
<td></td>
<td>CWE-114, Process Control</td>
</tr>
<tr>
<td></td>
<td>CWE-601, URL Redirection to Untrusted Site (‘Open Redirect’)</td>
</tr>
</tbody>
</table>

Bibliography

[Jack 2007]
[van Sprundel 2006]