EXP16-C. Do not compare function pointers to constant values

Comparing a function pointer to a value that is not a null function pointer of the same type will be diagnosed because it typically indicates programmer error and can result in unexpected behavior. Implicit comparisons will be diagnosed, as well.

Noncompliant Code Example

In this noncompliant code example, the addresses of the POSIX functions getuid and geteuid are compared for equality to 0. Because no function address shall be null, the first subexpression will always evaluate to false (0), and the second subexpression always to true (nonzero). Consequently, the entire expression will always evaluate to true, leading to a potential security vulnerability.

```c
/* First the options that are allowed only for root */
if (getuid == 0 || geteuid != 0) {
    /* ... */
}
```

Noncompliant Code Example

In this noncompliant code example, the function pointers getuid and geteuid are compared to 0. This example is from an actual vulnerability (VU#837857) discovered in some versions of the X Window System server. The vulnerability exists because the programmer neglected to provide the open and close parentheses following the geteuid() function identifier. As a result, the geteuid token returns the address of the function, which is never equal to 0. Consequently, the or condition of this if statement is always true, and access is provided to the protected block for all users. Many compilers issue a warning noting such pointless expressions. Therefore, this coding error is normally detected by adherence to MSC00-C. Compile cleanly at high warning levels.

```c
/* First the options that are allowed only for root */
if (getuid() == 0 || geteuid != 0) {
    /* ... */
}
```

Compliant Solution

The solution is to provide the open and close parentheses following the geteuid token so that the function is properly invoked:

```c
/* First the options that are allowed only for root */
if (getuid() == 0 || geteuid() != 0) {
    /* ... */
}
```

Compliant Solution

A function pointer can be compared to a null function pointer of the same type:

```c
/* First the options that are allowed only for root */
if (getuid == (uid_t(*)(void))0 || geteuid != (uid_t(*)(void))0) {
    /* ... */
}
```

This code should not be diagnosed by an analyzer.

Noncompliant Code Example

In this noncompliant code example, the function pointer do_xyz is implicitly compared unequal to 0:
int do_xyz(void);

int f(void) {
    /* ... */
    if (do_xyz) {
        return -1; /* Indicate failure */
    }
    /* ... */
    return 0;
}

Compliant Solution

In this compliant solution, the function do_xyz() is invoked and the return value is compared to 0:

int do_xyz(void);

int f(void) {
    /* ... */
    if (do_xyz()) {
        return -1; /* Indicate failure */
    }
    /* ... */
    return 0;
}

Risk Assessment

Errors of omission can result in unintended program flow.

<table>
<thead>
<tr>
<th>Recommendation</th>
<th>Severity</th>
<th>Likelihood</th>
<th>Remediation Cost</th>
<th>Priority</th>
<th>Level</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXP16-C</td>
<td>Low</td>
<td>Likely</td>
<td>Medium</td>
<td>P6</td>
<td>L2</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>Astrée</td>
<td>19.04</td>
<td>function-name-constant-comparison</td>
<td>Partially checked</td>
</tr>
<tr>
<td>Coverity</td>
<td>2017.07</td>
<td>BAD_COMPARE</td>
<td>Can detect the specific instance where the address of a function is compared against 0, such as in the case of geteuid versus getuid() in the implementation-specific details</td>
</tr>
<tr>
<td>GCC</td>
<td>4.3.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>CWARN.NULLCHECK.FUNCNAME, CWARN.FUNCADDR</td>
<td>Can detect violations of this recommendation when the -Wall flag is used</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>99 S</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-EXP16-a</td>
<td>Function address should not be compared to zero</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>0428, 3004, 3344</td>
<td></td>
</tr>
<tr>
<td>PVS-Studio</td>
<td>7.07</td>
<td>V516</td>
<td></td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>function-name-constant-comparison</td>
<td>Partially checked</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>Standard</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEI CERT C++ Coding Standard</td>
<td>VOID EXP16-CPP. Avoid conversions using void pointers</td>
</tr>
<tr>
<td>ISO/IEC TR 24772:2013</td>
<td>Likely incorrect expressions [KOA]</td>
</tr>
<tr>
<td>ISO/IEC TS 17961</td>
<td>Comparing function addresses to zero [funcaddr]</td>
</tr>
<tr>
<td>MITRE CWE</td>
<td>CWE-480, Use of incorrect operator</td>
</tr>
<tr>
<td></td>
<td>CWE-482, Comparing instead of assigning</td>
</tr>
</tbody>
</table>

### Bibliography

[Hatton 1995] Section 2.7.2, "Errors of Omission and Addition"