**STR05-C. Use pointers to const when referring to string literals**

The type of a narrow string literal is an array of `char`, and the type of a wide string literal is an array of `wchar_t`. However, string literals (of both types) are notionally constant and should consequently be protected by `const` qualification. This recommendation is a specialization of DCL00-C. Const-qualify immutable objects and also supports STR30-C. Do not attempt to modify string literals.

Adding `const` qualification may propagate through a program as `const` qualifiers are added, still more become necessary. This phenomenon is sometimes called `const-poisoning`. `Const-poisoning` can frequently lead to violations of EXP05-C. Do not cast away a `const` qualification. Although `const` qualification is a good idea, the costs may outweigh the value in the remediation of existing code.

**Noncompliant Code Example (Narrow String Literal)**

In this noncompliant code example, the `const` keyword has been omitted:

```c
char *c = "Hello";
```

If a statement such as `c[0] = 'C'` were placed following the declaration in the noncompliant code example, the code is likely to compile cleanly, but the result of the assignment would be `undefined` because string literals are considered constant.

**Compliant Solution (Immutable Strings)**

In this compliant solution, the characters referred to by the pointer `c` are `const`-qualified, meaning that any attempt to assign them to different values is an error:

```c
const char *c = "Hello";
```

**Compliant Solution (Mutable Strings)**

In cases where the string is meant to be modified, use initialization instead of assignment. In this compliant solution, `c` is a modifiable `char` array that has been initialized using the contents of the corresponding string literal:

```c
char c[] = "Hello";
```

Consequently, a statement such as `c[0] = 'C'` is valid and behaves as expected.

**Noncompliant Code Example (Wide String Literal)**

In this noncompliant code example, the `const` keyword has been omitted:

```c
wchar_t *c = L"Hello";
```

If a statement such as `c[0] = L'C'` were placed following this declaration, the code is likely to compile cleanly, but the result of the assignment would be `undefined` because string literals are considered constant.

**Compliant Solution (Immutable Strings)**

In this compliant solution, the characters referred to by the pointer `c` are `const`-qualified, meaning that any attempt to assign them to different values is an error:

```c
wchar_t const *c = L"Hello";
```

**Compliant Solution (Mutable Strings)**

In cases where the string is meant to be modified, use initialization instead of assignment. In this compliant solution, `c` is a modifiable `wchar_t` array that has been initialized using the contents of the corresponding string literal:

```c
wchar_t c[] = L"Hello";
```
wchar_t c[] = L"Hello";

Consequently, a statement such as c[0] = L'C' is valid and behaves as expected.

Risk Assessment

Modifying string literals causes **undefined behavior**, resulting in **abnormal program termination** and **denial-of-service vulnerabilities**.

<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>STR05-C</td>
<td>Low</td>
<td>Unlikely</td>
<td>Low</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>Astrée</td>
<td>19.04</td>
<td>literal-assignment</td>
<td>Fully checked</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-STR05</td>
<td></td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.2p0</td>
<td>LANG.TYPE.NCS</td>
<td>Non-const string literal</td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.STR05</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Klocwork</td>
<td>2018</td>
<td>MISRA.STRING_LITERAL.NON_CONST.2012</td>
<td></td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>9.7.1</td>
<td>623 S</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>Parasoft C/C++ test</td>
<td>10.4.2</td>
<td>CERT_C-STR05-a</td>
<td>A string literal shall not be modified</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>0752 0753</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>RuleChecker</td>
<td>19.04</td>
<td>literal-assignment</td>
<td>Fully checked</td>
</tr>
</tbody>
</table>

Related Vulnerabilities

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

Bibliography

[Corfield 1993]

[Lockheed Martin 2005] AV Rule 151.1