STR11-C. Do not specify the bound of a character array initialized with a string literal

The C Standard allows an array variable to be declared both with a bound index and with an initialization literal. The initialization literal also implies an array size in the number of elements specified. For strings, the size specified by a string literal is the number of characters in the literal plus one for the terminating null character.

It is common for an array variable to be initialized by a string literal and declared with an explicit bound that matches the number of characters in the string literal. Subclause 6.7.9, paragraph 14, of the C Standard [ISO/IEC 9899:2011], says:

> An array of character type may be initialized by a character string literal or UTF8 string literal, optionally enclosed in braces. Successive bytes of the string literal (including the terminating null character if there is room or if the array is of unknown size) initialize the elements of the array.

However, if the string is intended to be used as a null-terminated byte string, then the array will have one too few characters to hold the string because it does not account for the terminating null character. Such a sequence of characters has limited utility and has the potential to cause vulnerabilities if a null-terminated byte string is assumed.

A better approach is to not specify the bound of a string initialized with a string literal because the compiler will automatically allocate sufficient space for the entire string literal, including the terminating null character. This rule is a specific exception to ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer.

Noncompliant Code Example

This noncompliant code example initializes an array of characters using a string literal that defines one character more (counting the terminating ‘\0’) than the array can hold:

```c
const char s[3] = "abc";
```

The size of the array `s` is 3, although the size of the string literal is 4. Any subsequent use of the array as a null-terminated byte string can result in a vulnerability, because `s` is not properly null-terminated. (See STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string.)

Implementation Details

This code compiles with no warning with Visual Studio 2013 and GCC 4.8.1. It produces a three-character array with no terminating null character, as specified by the standard.

Compliant Solution

This compliant solution does not specify the bound of a character array in the array declaration. If the array bound is omitted, the compiler allocates sufficient storage to store the entire string literal, including the terminating null character.

```c
const char s[] = "abc";
```

This approach is preferred because the size of the array can always be derived even if the size of the string literal changes.

Exceptions

STR11-C-EX1: If the intention is to create a character array and not a null-terminated byte string, initializing to fit exactly without a null byte is allowed but not recommended. The preferred approach to create an array containing just the three characters ‘a’, ‘b’, and ‘c’, for example, is to declare each character literal as a separate element as follows:

```c
char s[3] = { 'a', 'b', 'c' }; /* NOT a string */
```

Also, you should make clear in comments or documentation if a character array is, in fact, not a null-terminated byte string.

STR11-C-EX2: If the character array must be larger than the string literal it is initialized with, you may explicitly specify an array bounds. This is particularly important if the array’s contents might change during program execution.
```c
#include <string.h>

void func(void) {
    char s[10] = "abc";
    strcpy(s[3], "def");
}
```

### Risk Assessment

<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>STR11-C</td>
<td>Low</td>
<td>Probable</td>
<td>Low</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>CertC-STR11</td>
<td>Supported: Astrée can detect subsequent code defects that this rule aims to prevent.</td>
</tr>
<tr>
<td>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-STR11</td>
<td></td>
</tr>
<tr>
<td>Compass/ROSE</td>
<td>9.7.1</td>
<td>404 S</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>ECLAIR</td>
<td>1.2</td>
<td>CC2.STR36</td>
<td>Fully implemented</td>
</tr>
<tr>
<td>LDRA tool suite</td>
<td>10.4.2</td>
<td>CERT_C-STR11-a</td>
<td>Do not specify the bound of a character array initialized with a string literal</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2019b</td>
<td>CERT_C: Rec. STR11-C</td>
<td>Checks for missing null in string array (rec. partially covered)</td>
</tr>
<tr>
<td>PRQA QA-C</td>
<td>9.7</td>
<td>1312</td>
<td>Partially implemented</td>
</tr>
<tr>
<td>Splint</td>
<td>3.1.1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Related Vulnerabilities

Search for vulnerabilities resulting from the violation of this rule on the [CERT website](https://www.cert.org).

### Related Guidelines

- **CERT C Secure Coding Standard**: ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer
- **CERT C Secure Coding Standard**: STR32-C. Do not pass a non-null-terminated character sequence to a library function that expects a string
- **SEI CERT C++ Coding Standard**: VOID STR08-CPP. Do not specify the bound of a character array initialized with a string literal
- **ISO/IEC TR 24772:2013**: String Termination [CJM]

### Bibliography

- [ECTC 1998](#) Section A.8, "Character Array Initialization"
- [ISO/IEC 9899:2011](#) Subclause 6.7.9, "Initialization"
- [Seacord 2013](#) Chapter 2, "Strings"