STR07-C. Use the bounds-checking interfaces for string manipulation

The C Standard, Annex K (normative), defines alternative versions of standard string-handling functions designed to be safer replacements for existing functions. For example, it defines the `strcpy_s()`, `strcat_s()`, `strncpy_s()`, and `strncat_s()` functions as replacements for `strcpy()`, `strcat()`, `strncpy()`, and `strncat()`, respectively.

The Annex K functions were created by Microsoft to help retrofit its existing legacy code base in response to numerous, well-publicized security incidents over the past decade. These functions were subsequently proposed to the international standardization working group for the programming language C (ISO/IEC JTC1/SC22/WG14) for standardization.

The `strcpy_s()` function, for example, has this signature:

```c
errno_t strcpy_s(char * restrict s1, rsize_t s1max, const char * restrict s2);
```

The signature is similar to `strcpy()` but takes an extra argument of type `rsize_t` that specifies the maximum length of the destination buffer. Functions that accept parameters of type `rsize_t` diagnose a constraint violation if the values of those parameters are greater than `RSIZE_MAX`. Extremely large object sizes are frequently a sign that an object’s size was calculated incorrectly. For example, negative numbers appear as very large positive numbers when converted to an unsigned type like `size_t`. For those reasons, it is sometimes beneficial to restrict the range of object sizes to detect errors. For machines with large address spaces, the C Standard, Annex K, recommends that `RSIZE_MAX` be defined as the smaller of the size of the largest object supported or `(SIZE_MAX >> 1)`, even if this limit is smaller than the size of some legitimate, but very large, objects (see also INT01-C. Use `size_t` or `rsize_t` for all integer values representing the size of an object).

The semantics of `strcpy_s()` are similar to the semantics of `strcpy()`. When there are no input validation errors, the `strcpy_s()` function copies characters from a source string to a destination character array up to and including the terminating null character. The function returns 0 on success.

The `strcpy_s()` function succeeds only when the source string can be fully copied to the destination without overflowing the destination buffer. Specifically, the following checks are made:

- The source and destination pointers are checked to see if they are `NULL`.
- The maximum length of the destination buffer is checked to see if it is equal to 0, greater than `RSIZE_MAX`, or less than or equal to the length of the source string.
- Copying is not allowed between objects that overlap.

When a runtime-constraint violation is detected, the destination string is set to the null string (as long as it is not a null pointer, and the maximum length of the destination buffer is greater than 0 and not greater than `RSIZE_MAX`), and the function returns a nonzero value. In the following example, the `strcpy_s()` function is used to copy `src1` to `dst1`:

```c
char src1[100] = "hello";
char src2[8] = {'g','o','o','d','b','y','e','\0'};
char dst1[6];
char dst2[5];
int r1;
int r2;

r1 = strcpy_s(dst1, sizeof(dst1), src1);
r2 = strcpy_s(dst2, sizeof(dst2), src2);
```

However, the call to copy `src2` to `dst2` fails because insufficient space is available to copy the entire string, which consists of eight characters, to the destination buffer. As a result, `r2` is assigned a nonzero value and `dst2[0]` is set to the null character.

Users of the C Standard Annex K functions are less likely to introduce a security flaw because the size of the destination buffer and the maximum number of characters to append must be specified. ISO/IEC TR 24731 Part II [ISO/IEC TR 24731-2:2010] offers another approach, supplying functions that allocate enough memory for their results. ISO/IEC TR 24731 Part II functions also ensure null termination of the destination string.

The C Standard Annex K functions are still capable of overflowing a buffer if the maximum length of the destination buffer and number of characters to copy are incorrectly specified. ISO/IEC TR 24731 Part II functions can make it more difficult to keep track of memory that must be freed, leading to memory leaks. As a result, the C Standard Annex K and the ISO/IEC TR 24731 Part II functions are not particularly secure but may be useful in preventive maintenance to reduce the likelihood of vulnerabilities in an existing legacy code base.

Noncompliant Code Example

This noncompliant code overflows its buffer if `msg` is too long, and it has undefined behavior if `msg` is a null pointer:
void complain(const char *msg) {
    static const char prefix[] = "Error: ";
    static const char suffix[] = "\n";
    char buf[BUFSIZ];
    strcpy(buf, prefix);
    strcat(buf, msg);
    strcat(buf, suffix);
    fputs(buf, stderr);
}

Compliant Solution (Runtime)
This compliant solution will not overflow its buffer:

void complain(const char *msg) {
    errno_t err;
    static const char prefix[] = "Error: ";
    static const char suffix[] = "\n";
    char buf[BUFSIZ];
    err = strcpy_s(buf, sizeof(buf), prefix);
    if (err != 0) { /* Handle error */
    }
    err = strcat_s(buf, sizeof(buf), msg);
    if (err != 0) { /* Handle error */
    }
    err = strcat_s(buf, sizeof(buf), suffix);
    if (err != 0) { /* Handle error */
    }
    fputs(buf, stderr);
}

Compliant Solution (Partial Compile Time)
This compliant solution performs some of the checking at compile time using a static assertion (see DCL03-C. Use a static assertion to test the value of a constant expression).
void complain(const char *msg) {
    errno_t err;
    static const char prefix[] = "Error: ";
    static const char suffix[] = "\n";
    char buf[BUFSIZ];

    /*
    * Ensure that more than one character
    * is available for msg
    */
    static_assert(sizeof(buf) > sizeof(prefix) + sizeof(suffix),
                "Buffer for complain() is too small");
    strcpy(buf, prefix);
    err = strcat_s(buf, sizeof(buf), msg);
    if (err != 0) {
        /* Handle error */
    }

    err = strcat_s(buf, sizeof(buf), suffix);
    if (err != 0) {
        /* Handle error */
    }
    fputs(buf, stderr);
}

Risk Assessment

String-handling functions defined in the C Standard, subclause 7.24, and elsewhere are susceptible to common programming errors that can lead to serious, exploitable vulnerabilities. Proper use of the C11 Annex K functions can eliminate most of these issues.

<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>STR07-C</td>
<td>High</td>
<td>Probable</td>
<td>Medium</td>
<td>P12</td>
<td>L1</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>Axivion Bauhaus Suite</td>
<td>6.9.0</td>
<td>CertC-STR07</td>
<td>Use of OemToAnsi, use of OemToChar (both include checks for uses of similar functions)</td>
</tr>
<tr>
<td>CodeSonar</td>
<td>5.3p0</td>
<td>BADFUNC.BO.OEMTOCHAR</td>
<td>Use of strcat (includes checks for uses of similar library functions such as StrCatA(), wcsca t(), etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCAT</td>
<td>Use of strcatChainW</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCHR</td>
<td>Use of strchr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCMP</td>
<td>Use of strcmp (includes checks for uses of similar functions such as lstrcmp())</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCOLL</td>
<td>Use of strcoll</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCPY</td>
<td>Use of strcpy (includes checks for uses of similar library functions such as StrCCpy(), wcsncpy(), etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRCSPN</td>
<td>Use of strstrpbrk</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRLEN</td>
<td>Use of strlen (includes checks for uses of similar functions such as lstrlen())</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRPBRK</td>
<td>Use of strstrchr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRSPN</td>
<td>Use of strspn</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRSTR</td>
<td>Use of strstr</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRTOK</td>
<td>Use of strtok</td>
</tr>
<tr>
<td></td>
<td></td>
<td>BADFUNC.BO.STRTRNS</td>
<td>Use of strtrns</td>
</tr>
</tbody>
</table>

<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>44 S</td>
<td>Enhanced enforcement</td>
</tr>
<tr>
<td>Parasoft C/C++test</td>
<td>10.4.2</td>
<td>CERT_C-STR07-a</td>
<td>Avoid using unsafe string functions that do not check bounds</td>
</tr>
<tr>
<td>Parasoft Insure++</td>
<td></td>
<td></td>
<td>Runtime analysis</td>
</tr>
<tr>
<td>Polyspace Bug Finder</td>
<td>R2020a</td>
<td>CERT C: Rec. STR07-C</td>
<td>Checks for:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Use of dangerous standard function</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Destination buffer overflow in string manipulation</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Rec. partially covered.</td>
</tr>
<tr>
<td>PROQA QA-C</td>
<td>9.7</td>
<td>5008</td>
<td>Partially implemented</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>ISO/IEC TR 24772:2013</td>
<td>Use of Libraries [TRJ]</td>
</tr>
</tbody>
</table>

**Bibliography**

<table>
<thead>
<tr>
<th>Reference</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>[Seacord 2005b]</td>
<td>&quot;Managed String Library for C, C++&quot;</td>
</tr>
<tr>
<td>[Seacord 2013]</td>
<td>Chapter 2, &quot;Strings&quot;</td>
</tr>
</tbody>
</table>