Security TR Editor's Report

December 1, 2004

Randy Meyers

1. Introduction

Let me apologize for the latest draft of the Security TR (N1088) being a week and a half late. I developed a nasty respiratory infection that required over twenty days of antibiotics to shake.

Because the TR was running late and the time schedule for review was tight, after consulting with John Benito, some of the new functions discussed in Redmond were not included in the draft. This was merely due to time pressure, and not a statement about the merits of those functions. The upcoming ISO registration of the TR is not a functionality freeze, and so these functions are expected to go the next draft ***after*** the ISO registration ballot (the TR is frozen during balloting) . I will write up a complete description of these functions for the Wiki and put them in the pre-meeting mailing.

2. What's not in N1088?

The following functions left out of N1088 because of time pressure are:

2.1 printf_s family (from <stdio.h> and <wchar.h>)

from <stdio.h>: `fprintf_s, printf_s, snprintf_s, sprintf, vfprintf_s, vprintf_s, vsnprintf_s, vsprintf_s`

from <wchar.h>: `fwprintf_s, swprintf_s, vfwprintf_s, vwprintf_s, wprintf_s, vswprintf_s`

All these printf functions have the same prototypes as their non-**`_s`** counterparts, except for `sprintf_s` and `vsprintf_s`, which match the prototypes for `snprintf` and `vsnprintf`, respectively.

These all of these functions differ from the non-`_s` counterparts by not supporting the `%n` format conversion specifier (a security risk) and by making it diagnosable undefined behavior if pointers are null or the format string is invalid.

There is a possibility that `sprintf_s` (and related forms) might also share the ability of `snprintf_s` (and related forms) to tell you the number of characters needed if the destination string is too small.

2.2 From <stdlib.h>

```
errno_t mbstowcs_s(size_t *restrict retval,
     wchar_t *restrict pwcs, rsize_t pwcsmax,
     const char *restrict s, rsize_t n);

errno_t wcstombs_s(size_t *restrict retval,
     char *s, rsize_t smax,
     const wchar_t *pwcs, rsize_t n);
```

These functions are like their non-`_s` counterparts except that they now return an error code, the new parameter `retval` is the non-`_s` return value, and the output string pointer parameter is now followed by a "max" parameter giving the number of elements in the output array. Of course, the max parameter is used to prevent overwriting the end of the buffer.

2.3 From <wchar.h>

```
errno_t mbsrtowcs_s(size_t *restrict retval,
     wchar_t *restrict dst, rsize_t dstmax,
     const char **restrict src, rsize_t len,
     mbstate_t *restrict ps);

errno_t wcsrtombs_s(size_t *restrict retval,
     char *restrict dst, rsize_t dstmax,
     const wchar_t **restrict src, rsize_t len,
     mbstate_t *restrict ps);

errno_t wcrtomb_s(size_t *restrict retval,
     char *restrict s, size_t smax,
     wchar_t wc, mbstate_t * restrict ps);
```

These functions are like their non-`_s` counterparts except that they now return an error code, the new parameter `retval` is the non-`_s` return value, and the output string pointer parameter is now followed by a "`max`" parameter giving the number of elements in the output array. Of course, the max parameter is used to prevent overwriting the end of the buffer.

2.4 Special arguments to strncpy_s, etc

Microsoft defines a special macro:

```
#define _TRUNCATE (size_t)-1
```

That can be used as the fourth argument to strncpy_s. E.g.,

```
strncpy_s(dest, sizeof dest, verylongsrc, _TRUNCATE);
```

This tells `strncpy_s` to truncate `verylongsrc` to fit into `dest` if necessary (`dest` is always null terminated). If `strlen(verlongsrc) < sizeof dest`, and thus `verylongsrc` copied without truncation, `strncpy_s` returns zero. Alternatively, if `strlen(verlongsrc) >= sizeof dest`, and thus the string copied but was truncated, `strncpy_s` returns a non-zero `errno_t` status meaning the string copied but was truncated.

3.0 Changes in N1088

3.1 New functions

Added to <stdio.h>: tmpfile_s, fopen_s, freopen_s
Added to <stdlib.h>: wctomb_s

3.2 rand_s removed

As requested by the committee, the rand_s function and associated macros were removed from the TR. (Just too hard to say anything meaningful, and possibly too much of a burden to implement.)

3.3 scanf_s family

The scanf family no longer use precision specifiers in the format string to get the maximum size of arrays being read. Instead, as in earlier drafts of the TR, they simply require that a size argument follow an argument matching a `c`, `s`, or `[` conversion specifier in the format.

3.4 Use of __STDC_WANT_SECURE_LIB_ _

As requested by the committee, Subclause 5.1.1 Paragraph 4 now requires that the "want" macro be defined the same way anytime any secure header is included. You can no longer get the secure version of <stdio.h> but the plain version of <string.h> (at least within the same compilation unit).

3.5 Reserved names

As requested by the committee, Subclause 5.1.2 was added to state under what conditions secure names are reserved.

3.6 Diagnosed Undefined Behavior

Several of the changes requested by the committee merged into one complex, extensive edit of the document. Most of the change bars in the document arose from this edit.

First, there was the committee's desire to remove undefined behavior. The previous draft of the TR (N1078) contained many explicit instructions in many of the function descriptions about what to do if an argument was, for example, unexpectedly null. In that draft, such cases called for the function to return an error code.

My previous editor's report (N1079) raised that as an issue. Previous to that draft, such cases were simply undefined behavior, and Microsoft took advantage of the leeway that

allowed to diagnose such cases at runtime, and call user written handlers to handle the situation. The committee also knew of similar cases in C99 handled similarly by debugging environments. Turning those situations into error codes that might be ignored (out of laziness or ignorance) by the program was not an improvement.

In Redmond, the general issue of undefined behavior was discussed. The latitude given by undefined behavior which permits an implementation to allow a program quietly to corrupt memory and continue running was the very thing that the secure library was designed to prevent. To the extent possible, undefined behavior should be removed from the secure library. I was instructed to remove as much undefined behavior as possible.

The committee then endorsed an idea tentatively called "secure constrained behavior," which would permit an implementation to diagnose and stop a program that does something bad, and would require the implementation to prevent a program from quietly corrupting memory and continuing to run. Returning a failure code rather than overwriting memory in strncpy_s (for example) was seen as an acceptable solution, as was aborting the program or failing an assert.

Along with this, the committee approved the concept of a range limited version of `size_t`, called `rsize_t`. `rsize_t` values larger than `RSIZE_MAX` are suspect, and should be diagnosed by functions in the secure library. This was another case of "secure constrained behavior."

In this draft, the concept with the working title "secure constrained behavior" became "diagnosed undefined behavior," a phrase that reads better when used in function descriptions in the TR. Diagnosed undefined behavior is behavior that previously would have been just undefined behavior. But, under the new scheme, diagnosed undefined behavior must be diagnosed by the implementation and the implementation will act in a benign way afterwards. From a programmer's point of view, all undefined behavior, like dereferencing a null pointer, is bad, but in some contexts, it will be diagnosed. Hence, the name.

Subclause 3.1 defines diagnosed undefined behavior and explains the model. I strongly urge you to read that definition to get the details, but a short summary of the model is this: Parts of the TR state that certain conditions are diagnosed undefined behavior. (For example, a null pointer for the format string to scanf_s.) The implementation must check for any such condition, and "diagnose" it by calling an implementation-defined function.

That implementation-defined function need not do anything. It might just return. It might halt the program. It might be a failing assert. If that implementation-defined function returns, then the implementation must behave according to the rules associated with the particular condition of diagnosed undefined behavior. For example, `scanf_s` says that if there is diagnosed undefined behavior, `scanf_s` does not attempt any input. Finally, the "Returns" section in the write up of a function usually gives a special return value if there is diagnosed undefined behavior. For example, `scanf_s` returns `EOF` if there is diagnosed undefined behavior.

This model permits implementations to diagnose dangerous conditions and halt the program if necessary (or call a handler, or what have you). The model requires that implementations behave gracefully in the presence of such conditions, and to return an error code rather than corrupting memory if they allow programs to run after such conditions are detected.

In terms of edits, almost every function description has a new first paragraph listing the diagnosed undefined behaviors for that function. The "Returns" clause usually has a short statement about what should be returned if undefined behaviors occur.

Making this edit sharpened a distinction in my mind. There are really two types of error conditions in programs. The first are errors that should never occur in a properly written program (e.g., dereferencing a null pointer). The second is problems that are unfortunate but a properly written program should be prepared to deal with if they occur (e.g., trying to fopen a missing file). The first category is a good candidate for diagnosed undefined behavior. The second category should not be diagnosed undefined behavior. Care should always be taken not to incorrectly make something diagnosed undefined behavior.

3.7 `rsize_t`

As discussed above, input `size_t` parameters to functions were changed to have `rsize_t` type, and it is diagnosed undefined behavior if the values of such parameters exceed `RSIZE_MAX` (defined in Subclause 5.3).

3.8 Return Values

As discussed in committee, many implementations have a far more interesting set of errno values than in the Standard, and they would like to use those values as return codes from the secure library.

As requested by the committee, I changed functions that return an errno_t value to return zero for success, and non-zero for failure.

4. Other issues

4.1 strnlen_s, wcsnlen_s

I did not change the size_t parameters to these two functions to rsize_t. My reason is that these functions are most useful in dodgy circumstances, and another layer of checking in them might get in the way.

On the other hand, I am lightheaded from all that coughing. This should be reconsidered at the next meeting.

4.2 `qsort_s`

The `qsort_s` function does have diagnosed undefined behavior, but it returns void. So, on implementations that choose not to halt the program when there is diagnosed

undefined behavior, the function simply does nothing and returns. Perhaps it should be changed to return an errno_t?