Introduction to N1031

Walk through, issues, and rationale

Components of N1031

- New functions that protect against buffer overflow and always produce null terminated strings
- New reentrant versions of old functions
- New random number generator for cryptography

Guiding Principles

- If safe alternative exists, don't create a new function
- Not a war against null terminated strings
  - Most functions trust that input string parameters are null terminated
  - Output string parameters get a length argument
- Allow for compile-time checking for good programming

Guiding Principles

- Where possible, have functions return a code indicating success or reason for failure
  - regularity
  - helps with compile-time checking
- Failed functions should produce output values that prohibit carrying on as if no error occurred
- Minimize effort to port to new library

Which headers?

- One header for all the new functions?
  - Messy, no functional grouping
- Parallel system of headers? Eg, <string_s.h>
  - Lots of useful functions from old header, so both old and new headers will be included
- Put new functions in header as old versions?
  - Natural, but namespace issues

Namespace issues

- 7.26 Future Library Directions
- Many of the names fit the patterns for names that can be added to the headers
  - str* to <string.h>
- Many of the names do not fit the patterns
  - *scanf to <stdio.h>
  - wmem* to <wchar.h>

Possibilities

- Add allowed names to headers and protect via a macro names not allowed

```
#ifdef __USE_SECURE_LIB__
int fscanf_s(FILE * restrict stream,
    const char * restrict format, ...);
#endif
```

Possibilities

- Or, protect all new names via macro
- Might minimize compatibility problems for “bad” programs that step on Standard namespace
- Easy rule to remember

Discussion/Straw Poll

- In favor of adding functions to existing headers?
- In favor of protecting all new functions via a macro
- Any better name for the `__USE_SECURE_LIB__` macro?

Return value

- Return an errno value
- zero is success
- ERANGE used to indicate output buffer too small
- Precedent from Single Unix Spec
- E2BIG as an alternative to ERANGE?

`int` verses typedef

- `int` is a rather bland type
- Could have

```
typedef int errcode_t;
errcode_t strncpy_s(
    char *restrict s1,
    size_t s1max,
    const char *restrict s2,
    size_t n);
```

Discussion/Straw Poll

- In favor in principle to making the return value an indication of success / failure?
- In favor in principle to making the return value an errno value?
- In favor using `errno_t` as the return type when functions return an errno value?

scanf_s family

- Considered using maximum field width to express capacity of receiving variable
- Insufficient for wscanf family where “%22s” means input a token of up to 22 wide chars and store into a multibyte string of locale-dependent size

rand_s()

- Expect better description next draft
- Might exploit hardware random number generators
- Might lack
  - user specified seed
  - restarting a sequence of random numbers

strncpy_s

```
strncpy_s(a, sizeof a, b, sizeof b);
```

- succeeds if and only if a null terminated string from "b" fits in "a". In this form, strncpy_s is equivalent to a safe version of strcpy.

strncpy_s

- If you don't know the actual size of the array b but you trust that it is either null terminated or has a size greater than sizeof a, you can make the call:

```
strncpy_s(a, sizeof a, b, sizeof a);
```

strncpy_s

- If you want a truncating version of strncpy_s, and you trust that b is either null terminated or has a size greater than sizeof a, you can make a call like:

```
strncpy_s(a, sizeof a, b, (sizeof a)-1);
```

strncpy_s

- Paragraph 5 allows for efficient copy
- Paragraph 5 also allows for strncpy()-like null padding
- Paragraph 5 probably should become a global statement about any string result

Discussion/Straw Poll

- In favor of the license given paragraph 5?
- In favor of making paragraph 5 apply to string results from other functions when a bound for the output array is known?
- Recommend for or against null padding like strncpy?

Programming Practices Annex

- Should the TR have an informational Annex listing functions to be avoided in favor of new functions?

Implementation Issues

- Should the Security TR contain sections addressing quality of implementation issues like parameter validation for old functions, checks for NULL pointers, etc?

Rationale

- Should Rationale be provided for the TR?
- Interspersed or parallel document or Annex?

Proposals for next draft

Feature Macro

- Predefined macro indicating library is available?

New Functions

```
int strcpy_s(char *restrict s1, size_t s1max, const char
    *restrict s2);
int strcat_s(char *restrict s1, size_t s1max, const char *restrict
    s2);
int wcscpy_s(wchar_t *restrict s1, size_t s1max, const
    wchar_t *restrict s2);
int wcscat_s(wchar_t *restrict s1, size_t s1max, const wchar_t
    *restrict s2);
```

Failing scanf_s

- Any variables not successfully read into by scanf_s should be set to values designed to prevent accidental uses of those variables.

scanf_s("%s %s", a, sizeof a, b, sizeof b);

- If scanf_s returns 1 because EOF prevented reading `b`, then `b[0]` should be set to '\0'

Unix Compatibility

- In some cases, these functions were inspired by or similar to functions in the Single Unix Spec.
- A careful comparison with the Single Unix Spec will accompany the next draft