Security and Standard C Libraries

Martyn Lovell
Visual C++ Libraries
Microsoft Corporation

Agenda

- Current Situation
- Security Problems
- Library impact
- Proposed Changes
- Related C++ issues
- Summary
- Q&A

Libraries Background

- Standard C Library initially evolved with Unix™
  - Starting in 1970s
  - Public networking uncommon
  - Machines simpler, more constrained
- First C language/library standard codified existing practice
  - High degree of compatibility with current use
  - Core set of required functions for C programmers

Current situation

- Public networking extremely common
- Motivated attackers common
  - Direct hacking
  - Social engineering (malicious spam)
- Firewalls not enough to protect network software
  - Laptops move in and out of firewall
  - Wireless networks allow promiscuous binding

Security implications

- Security is now critical for the whole software industry
- Much code that was previously unimportant now vulnerable to attack
- Assume every software component might be attacked
- Security issues require urgent attention
  - Software vendors issuing patches regularly
  - Locked-down customers need to deploy updates or workarounds

Improving the Situation

- Short Term
  - Be responsive
  - Have great security process
- Medium Term
  - Threat modelling
  - Security review
  - Defense in depth
  - Defaulting to security

Improving the Situation

- Long term
  - Development improvements
    - Better libraries
    - Better coding practices
    - Better development standards
    - Better trained developers
    - Better quality assurance
    - No single panacea
  - Standard C Libraries a core part of this plan
    - Need to make breaking changes
    - Help drive improvements across all deployed code

Key security problems

- Buffer overruns
- Error reporting
- Parameter validation
- File access rights
- Static buffer results

Security issues

- Standard-defined interface problems
  - Function shape is dangerous
  - New function needed
- Standard-defined implementation problems
  - Function requirements are dangerous
  - Existing function can change
- Standard-agnostic implementation problems
  - Standard change not required, may be useful

Standard Interface problems

- Lack of buffer size
  - fgets
- Lack of error return
  - Continue after error can be attackable
- Callback context needed
  - Avoid static variables
  - Safe for reentrancy, threads
  - qsort, bsearch

Standard interface problems

- Never use static result buffers
  - Can overrun
  - tmpnam
- Random number quality
  - Can allow predictable attacks
  - New name since different performance
  - Rand

Standard implementation problems

- Returning unterminated strings
  - Predictable, but hard to get right
  - Strncpy

Quality of Implementation issues

- Parameter validation
  - Ensure errors reported
  - Provide way to catch errors
    - We will have handler function
  - assert in "debug" build
- Stack usage
  - Avoid excessive usage
  - Stack overflow can be a denial of service attack

Quality of Implementation issues

- File permissions
  - Create temporary files safely
  - Create all files by default with good permissions
- Scanf
  - No totally safe way to implement this
  - Consider requiring buffer sizes
- Long file path support
  - Don't fail when given extended paths
  - Windows specific

Our Secure Libraries

- Reviewing all 2000 C and C++ library functions
  - Includes many MS extensions
- Creating around 400 secure variants
- Variants follow secure coding rules
- Will not make an application secure
  - But will help build one
  - Removes need for user to build secure replacements

Our secure libraries

- Give the developer option to deprecate insecure functions
  - Deprecate by default
    - __declspec(deprecate) in VC++
    - Causes a compiler warning
  - Allow deprecation to be removed
    - Using a #define
  - Makes us secure by default

Example

- **Old:**
  - size_t mbstowcs
    (
     wchar_t *wcstr,
     const char *mbstr,
     size_t count
    );
- New:
  - **errcode** mbstowcs_s
    (
     size_t *pConvertedMBChars,
     wchar_t *wcstr,
     size_t sizeInWords,
     const char *mbstr,
     size_t count
    );

Scope of changes

- Most functions change implementation
  - For validation
  - For quality of implementation issues
- 37 functions have some interface shape change
  - Not time to review them all here
  - Details in my paper proposal
  - List on next page

Functions changing

- bsearch, qsort
  - Pass context
- tmpnam, getenv, strtok, ctime
  - Don't use static state
- rand
  - Crypto-safe
- fgets, gets, vsprintf, strerror, strncat. Wcsncat, strncpy, wcsncpy, wcstombs, mbstowcs
  - Buffer size

Functions changing

- memcpy, wmemcpy, memmove, wmemmove
  - Separate dest size
- fscanf, scanf, wscanf, sscanf, swscanf
  - scanf family problems
- setbuf, sprintf, strcat, strcpy, wcscat, wcscpy
  - Deprecate
  - Better versions already in standard

Secure Libraries Questions

- Deprecate by default?
- Integer multiplication
- Naming convention
  - mbstowcs (old)
    - mbstowcs_s (new) OR
    - mbstowcsn (new)
- Change names always, or only for interface changes
  - Non-interface changes include more rigorous validation
- Error return
  - Provide alternative error mechanisms
- scanf family

C++ Library

- Early stages of planning
- Less changes required
  - Newer library
  - Less direct buffer manipulation
- Existing fixes
  - operator << into char []
    - Replace with alternative mechanism
  - Validation
  - Bounds checking
  - Quality of implementation issues
  - Other small changes

C++ Library (2)

- New classes
  - Secret buffer
    - Erases after use
  - alloca replacement
  - size-checked stack array
  - Overrun detection wrappers
  - Path handling
    - URL
    - Disk

Summary

- Time to make the C library secure
- Help developers move their code forward to secure practices
- Exemplar implementation being built now
  - Will be available for trialing/review soon
  - My dev team is building right now

Questions?

- Follow-up:
  - MartynL@microsoft.com