Security Development Lifecycle (SDL) Banned Function Calls

06/12/2012 • 10 minutes to read

In this article

Michael Howard
Principal Cyber-Security Consultant
Microsoft Corporation

June 2011

**This paper is derived from the book *The Security Development Lifecycle* , by Michael Howard and Steve Lipner, Microsoft Press, 2006.

Prohibiting the use of banned functions is a good way to remove a significant number of potential code vulnerabilities from C and C++ code. This practice is reflected in chapter 11 of *The Security Development Lifecycle* book, as well as in the SDL Process Guidance and Simplified Implementation of the Microsoft SDL whitepaper within the SDL Implementation phase.

When the C runtime library (CRT) was first created over three decades ago, the threats to computers were different; computers were not as interconnected as they are today, and attacks were not as prevalent. With this in mind, a subset of the C runtime library must be deprecated for new code and, over time, removed from earlier code. It's just too easy to get code wrong that uses these outdated functions and these types of errors lead to vulnerabilities. Even some of the classic replacement functions are prone to error too.

This list is the compiled library of known bad functions that should be removed to reduce vulnerabilities as part of your SDL practices. It is derived from experience with real-world security bugs and focuses primarily on functions that can lead to buffer overruns (*24 Deadly Sins of Software Development*; Howard, LeBlanc, and Viega 2009). Existing code must either replace the banned function with a more secure version or be re-architected so that the banned function is not used. For the functions marked as "recommended", please consider this a strong recommendation and evaluate the function against your own security requirements, elevating them to "required" as necessary. In any case, none of the listed banned functions should be used in new code.

Notes:

- The banned functions are either required or recommended based on:
  - The assessed risk of the function with consideration of the overall difficulty of calling the function in a correct and "safe" manner.
  - The availability of a safe replacement and the complexity of implementing a replacement.
- Some of the function names might be a little different, depending on whether the function takes ASCII, Unicode, _T (ASCII or Unicode), or multibyte chars. Some function names might include *A* or *W* at the end of the name. For example, the StrSafe StringCbCatEx function is also available as StringCbCatExW (Unicode) and StringCbCatExA (ASCII).
- Obviously, you cannot replace a banned function with another banned function. For example, replacing strcpy with strncpy is not valid because strncpy is banned, too.

Table 1. Banned string copy functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
strcpy, strcpyA, strcpyW, wcscpy, _tcscpy, _mbscpy, StrCpy, StrCpyA, StrCpyW, lstrcpy, lstrcpyA, lstrcpyW, _tccpy, _mbccpy, _ftcscpy, strncpy, wcsncpy, _tcsncpy, _mbsncpy, _mbsnbcpy, StrCpyN, StrCpyNA, StrCpyNW, StrNCpy, strcpynA, StrNCpyA, StrNCpyW, lstrcpyn, lstrcpynA, lstrcpynW	String*1Copy or String*CopyEx	strcpy_s

1 For StrSafe, * should be replaced with Cch (character count) or Cb (byte count)

Table 2. Banned string concatenation functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
strcat, strcatA, strcatW, wcscat, _tcscat, _mbscat, StrCat, StrCatA, StrCatW, lstrcat, lstrcatA, lstrcatW, StrCatBuff, StrCatBuffA, StrCatBuffW, StrCatChainW, _tccat, _mbccat, _ftcscat, strncat, wcsncat, _tcsncat, _mbsncat, _mbsnbcat, StrCatN, StrCatNA, StrCatNW, StrNCat, StrNCatA, StrNCatW, lstrncat, lstrcatnA, lstrcatnW, lstrcatn	String*Cat or String*CatEx	strcat_s

Table 3. Banned sprintf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
sprintfW, sprintfA, wsprintf, wsprintfW, wsprintfA, sprintf, swprintf, _stprintf, wvsprintf, wvsprintfA, wvsprintfW, vsprintf, _vstprintf, vswprintf *Recommended*: wnsprintf, wnsprintfA, wnsprintfW, _snwprintf, snprintf, sntprintf _vsnprintf, vsnprintf, _vsnwprintf, _vsntprintf, wvnsprintf, wvnsprintfA, wvnsprintfW	String*Printf or String*PrintfEx	sprintf_s

Table 4. Banned "n" sprintf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: _snwprintf, _snprintf, _sntprintf, nsprintf	String*Printf or String*PrintfEx	_snprintf_s or _snwprintf_s

Table 5. Banned variable argument sprintf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
wvsprintf, wvsprintfA, wvsprintfW, vsprintf, _vstprintf, vswprintf	String*VPrintf or String*VPrintfEx	_vstprintf_s

Table 6. Banned variable argument "n" sprintf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: _vsnprintf, _vsnwprintf, _vsntprintf, wvnsprintf, wvnsprintfA, wvnsprintfW	String*VPrintf or String*VPrintfEx	vsntprintf_s

Table 7. Banned "n" string copy functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
strncpy, wcsncpy, _tcsncpy, _mbsncpy, _mbsnbcpy, StrCpyN, StrCpyNA, StrCpyNW, StrNCpy, strcpynA, StrNCpyA, StrNCpyW, lstrcpyn, lstrcpynA, lstrcpynW, _fstrncpy	String*CopyN or String*CopyNEx	strncpy_s

Table 8. Banned "n" string concatenation functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement

Banned Functions	StrSafe Replacement	Safe CRT Replacement
strncat, wcsncat, _tcsncat, _mbsncat, _mbsnbcat, StrCatN, StrCatNA, StrCatNW, StrNCat, StrNCatA, StrNCatW, lstrncat, lstrcatnA, lstrcatnW, lstrcatn, _fstrncat	String*CatN or String*CatNEx	strncat_s

Developers frequently replace functions like strcpy with the counted "n" version, such as strncpy. This practice is not recommended. In our experience, the "n" functions are also hard to secure (Howard 2004), so we have banned their use in new code.

Table 9. Banned string tokenizing functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: strtok, _tcstok, wcstok, _mbstok	None	strtok_s

Table 10. Banned Makepath functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: makepath, _tmakepath, _makepath, _wmakepath	None	_makepath_s

Table 11. Banned Splitpath functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: _splitpath, _tsplitpath, _wsplitpath	None	_splitpath_s

Table 12. Banned scanf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: scanf, wscanf, _tscanf, sscanf, swscanf, _stscanf	None	sscanf_s

Table 13. Banned "n" scanf functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: snscanf, snwscanf, _sntscanf	None	_snscanf_s

Table 14. Banned numeric conversion functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: _itoa, _itow, _i64toa, _i64tow, _ui64toa, _ui64tot, _ui64tow, _ultoa, _ultot, _ultow	None	_itoa_s, _itow_s

Table 15. Banned gets functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
gets, _getts, _gettws	String*Gets	gets_s

Table 16. Banned IsBad* functions and replacements

Banned Functions	
IsBadWritePtr, IsBadHugeWritePtr, IsBadReadPtr,	These functions can mask errors, and there are no replacement functions. You should rewrite the code to avoid using these functions. If you need to avoid a crash, wrap your usage of the pointer with __try/__except. Doing this can easily hide bugs; you should do this only in areas where it is absolutely critical

IsBadHugeReadPtr, IsBadCodePtr, IsBadStringPtr	to avoid a crash (such as crash recovery code) and where you have a reasonable explanation for why the data you're looking at might be invalid. You should also not catch all exceptions, but only types that you know about. Catching all exceptions is just as bad as using IsBad*Ptr. For IsBadWritePtr, filling the destination buffer using memset is a preferred way to validate that output buffers are valid and large enough to hold the amount of space that the caller claims they provided.

Table 17. Banned OEM conversion functions and replacements

Banned Functions	Windows Replacement
Recommended: CharToOem, CharToOemA, CharToOemW, OemToChar, OemToCharA, OemToCharW, CharToOemBuffA, CharToOemBuffW	WideCharToMultiByte

Table 18. Banned stack dynamic memory allocation functions and replacements

Banned Functions	Windows Replacement
Recommended: alloca, _alloca	SafeAllocA

For critical applications, such as those accepting anonymous Internet connections, strlen must also be replaced:

Table 19. Banned string length functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement
Recommended: strlen, wcslen, _mbslen, _mbstrlen, StrLen, lstrlen	String*Length	strnlen_s

Table 20. Banned memory copy functions and replacements

Banned Functions	StrSafe Replacement	Safe CRT Replacement

Banned Functions	StrSafe Replacement	Safe CRT Replacement
memcpy, RtlCopyMemory, CopyMemory, wmemcpy	None	memcpy_s, wmemcpy_s

Table 21. Banned window messaging functions and replacements

Banned Functions	
Recommended: ChangeWindowMessageFilter	This function is not recommended because it has process-wide scope. You can use ChangeWindowMessageFilterEx to control access for specific windows, but give careful consideration to any message filtering changes.

Why the "n" Functions Are Banned

The classic C runtime, "n" functions (such as strncpy and strncat) are banned because they are so hard to call correctly. The authors have seen numerous errors calling these functions in an attempt to make code more secure. Note that we're not saying the replacements are perfect, but issues with the current "n" functions include non-null termination of overflowed buffers and no error returns on overflow.

The newer StrSafe and Safe CRT functions are more consistent on failure.

Important Caveat

Simply replacing a banned function call with a better replacement does not guarantee that the code is secure. It's possible to misuse the replacement function, most commonly by getting the destination buffer size wrong.
One way to make sure the function call is safe is to double check that the numberOfElements argument in a Safe CRT function call (such as strcpy_s) or the cchDest argument in a StrSafe function (such as StringCchCopy) is no larger than the destination buffer

size. A good way to do this is to use a _countof(buffer) call. Examples are shown later in this document.

Choosing StrSafe vs. Safe CRT

There is an overlap between these two sets of replacement C runtime functions. Which you choose depends on your specific situation; the following table should help you make the decision. In some cases, you might have little choice but to use one over the other; for example, if your code calls itoa, there is no replacement in StrSafe, but there is in Safe CRT. You would need to either code around the itoa call or use Safe CRT.

Table 22.

	StrSafe	**Safe CRT**
Distribution Method	Web (*http://msdn.microsoft.com*)	Microsoft Visual Studio 2005 or later
# Headers	One (StrSafe.h)	Numerous (various C runtime headers)
Library Version Available	Yes	Yes
Inline Version Available	Yes	No
Industry Standard	No	Not Yet (Secure C Lib Functions)
Kernel Mode	Yes	No
Return Type	HRESULT (user) or NTSTATUS (kernel)	Varies by function (errno_t)
Requires Code Changes	Yes	Yes

	StrSafe	Safe CRT
Main Focus	Buffer overrun issues	Various, including buffer overruns

Using StrSafe

To use StrSafe in your C or C++ code, simply add the following header:

```
#include "strsafe.h"
```

This will make the functions inline. If you want to use the library version, Strsafe.lib, add the following to your code:

```
#define STRSAFE_LIB
#include "strsafe.h"
```

Note that all the StrSafe functions include Rtl versions for kernel use.

StrSafe Example

The following code

```
void Function(char *s1, char *s2) {
```

```
    char temp[32];
    strcpy(temp,s1);
    strcat(temp,s2);
}
```

when converted to StrSafe might look like this:

Copy

```
HRESULT Function(char *s1, char *s2) {
    char temp[32];
    HRESULT hr = StringCchCopy(temp,_countof(temp), s1);
    if (FAILED(hr)) return hr;
    return StringCchCat(temp,_countof(temp), s2);
}
```

Using Safe CRT

The Safe CRT is included starting with Visual Studio 2005. When you compile code using this compiler, it will automatically warn you of the deprecated functions in the code. Also, in some cases, the compiler will change some function calls to safe function calls if the destination buffer size is known at compile time and CRT_SECURE_CPP_OVERLOAD_STANDARD_NAMES is #defined in the code.

For example, the following code

Copy

```
int main(int argc, char* argv[]) {
    char t[10];
       ...
```

```
    if (2==argc)
        strcpy(t,argv[1]);

       ...
    return 0;

}
```

is changed by the compiler to this:

Copy

```
int main(int argc, char* argv[]) {
    char t[10];
   ...
   if (2==argc)
       strcpy_s(t,_countof(t),argv[1]);

   ...
   return 0;
}
```

Safe CRT Example

The following code

Copy

```
void Function(char *s1, char *s2) {
    char temp[32];
    strcpy(temp,s1);
    strcat(temp,s2);
```

```
}
```

when converted to the Safe CRT might look like this:

Copy

```
errno_t Function(char *s1, char *s2) {
    char temp[32];
    errno_t err = strcpy_s(temp,_countof(temp), s1);
    if (!err) return err;
    return strcat_s(temp,_countof(temp), s2);
}
```

Other Replacements

If you are using C++, you should seriously consider using the std::string template class rather than manipulating buffers directly.

Many *nix variants, including OpenBSD and some Linux operating systems, include support for string copy replacements strlcpy and strlcat (Miller and de Raadt 1999).

Tools Support

The Visual Studio 2005 (and later) compiler has built-in deprecations for these functions; all C4996 compiler warnings should be investigated to make sure that the function in question is not on the preceding banned list. Also, look out for code that disables this warning, such as *#pragma warning(disable:4996)*.

ROI and Cost Impact

As stated earlier, removing banned functions is one way to reduce potential security bugs with very little engineering effort, as can be seen at the start of this document. Some Microsoft security bulletins would not have been necessary if banned functions had not been used, so the potential cost savings for an organization can be significant.

Metrics and Goals

The metric to track is the number of banned functions in former code and in new code. The quantity should be zero for new code and should follow a glide path down over time for former code.

References

(Howard, LeBlanc, and Viega 2009)

Howard, Michael, David LeBlanc, and John Viega. *24 Deadly Sins of Software Development*. New York, NY: McGraw-Hill, 2009. Chapter 5, "Buffer Overruns."

(Howard 2004)

Howard, Michael. "Buffer Overflow in Apache 1.3.xx fixed on Bugtraq—the evils of strncpy and strncat," http://blogs.msdn.com/michael_howard/archive/2004/10/29/249713.aspx , October 2004.

(Miller and de Raadt 1999)

Miller, Todd C., and Theo de Raadt. USENIX Annual Technical Conference, "strlcpy and strlcat – Consistent, Safe String Copy and Concatenation," http://www.usenix.org/events/usenix99/full_papers/millert/millert_html/index.html , June 1999.