## NAME

**strcat**, **strncat** - concatenate strings

## LIBRARY

Standard C Library (libc, -lc)

## SYNOPSIS

**#include <string.h>**

*char **
**strcat**(*char * restrict s, const char * restrict append*);

*char **
**strncat**(*char * restrict s, const char * restrict append, size_t count*);

## DESCRIPTION

The **strcat**() and **strncat**() functions append a copy of the null-terminated string *append* to the end of the null-terminated string *s*, then add a terminating ‘\0’. The string *s* must have sufficient space to hold the result. If *s* and *append* overlap, the results are undefined.

The **strncat**() function appends not more than *count* characters from *append*, and then adds a terminating ‘\0’. If *s* and *append* overlap, the results are undefined.

## RETURN VALUES

The **strcat**() and **strncat**() functions return the pointer *s*.

## SEE ALSO

bcopy(3), memccpy(3), memcpy(3), memmove(3), strcpy(3), strlcat(3), strlcpy(3), wcscat(3)

## STANDARDS

The **strcat**() and **strncat**() functions conform to ISO/IEC 9899:1990 ("ISO C90").

## HISTORY

The **strcat**() function first appeared in the Programmer’s Workbench (PWB/UNIX) and was ported to Version 7 AT&T UNIX; **strncat**() first appeared in Version 7 AT&T UNIX.

## SECURITY CONSIDERATIONS

The **strcat**() function is easily misused in a manner which enables malicious users to arbitrarily change a running program’s functionality through a buffer overflow attack.

Avoid using **strcat**(). Instead, use **strncat**() or **strlcat**() and ensure that no more characters are copied to the destination buffer than it can hold.

Note that **strncat**() can also be problematic. It may be a security concern for a string to be truncated at all. Since the truncated string will not be as long as the original, it may refer to a completely different resource and usage of the truncated resource could result in very incorrect behavior. Example:

```
void
foo(const char *arbitrary_string)
{
        char onstack[8];

#if defined(BAD)
        /*
         * This first strcat is bad behavior.  Do not use strcat!
         */
        (void)strcat(onstack, arbitrary_string);   /* BAD! */
#elif defined(BETTER)
        /*
         * The following two lines demonstrate better use of
         * strncat().
         */
        (void)strncat(onstack, arbitrary_string,
           sizeof(onstack) - strlen(onstack) - 1);
#elif defined(BEST)
        /*
         * These lines are even more robust due to testing for
         * truncation.
         */
        if (strlen(arbitrary_string) + 1 >
           sizeof(onstack) - strlen(onstack))
                err(1, "onstack would be truncated");
        (void)strncat(onstack, arbitrary_string,
           sizeof(onstack) - strlen(onstack) - 1);
#endif
}
```