NAME

sdoc - guide to adding security considerations sections to manual pages

DESCRIPTION

This document presents guidelines for adding security considerations sections to manual pages. It provides two typical examples.

The guidelines for writing FreeBSD manual pages in groff_mdoc(7) mandate that each manual page describing a feature of the FreeBSD system should contain a security considerations section describing what security requirements can be broken through the misuse of that feature. When writing these sections, authors should attempt to achieve a happy medium between two conflicting goals: brevity and completeness. On one hand, security consideration sections must not be too verbose, or busy readers might be dissuaded from reading them. On the other hand, security consideration sections must not be incomplete, or they will fail in their purpose of instructing the reader on how to avoid all insecure uses. This document provides guidelines for balancing brevity and completeness in the security consideration section for a given feature of the FreeBSD system.

Where to Start

Begin by listing those general security requirements that can be violated through the misuse of the feature. There are four classes of security requirements:

integrity(example: non-administrators should not modify system binaries),

confidentiality(example: non-administrators should not view the shadow password file),

availability(example: the web server should respond to client requests in a timely fashion), and

correctness(example: the ps program should provide exactly the process table information listing functionality described in its documentation - no more, no less.)

A good security considerations section should explain how the feature can be misused to violate each general security requirement in the list. Each explanation should be accompanied by instructions the reader should follow in order to avoid a violation. When referencing potential vulnerabilities described in the Secure Programming Practices manual page, sprog(7), likewise cross-reference that document rather than replicating information. Whenever possible, refer to this document rather than reproducing the material it contains.

Where to Stop

Security problems are often interrelated; individual problems often have far-reaching implications. For example, the correctness of virtually any dynamically-linked program is dependent on the correct

implementation and configuration of the run-time linker. The correctness of this program, in turn, depends on the correctness of its libraries, the compiler used to build it, the correctness of the preceding compiler that was used to build that compiler, and so on, as described by Thompson (see *SEE ALSO*, below).

Due to the need for brevity, security consideration sections should describe only those issues directly related to the feature that is the subject of the manual page. Refer to other manual pages rather than duplicating the material found there.

EXAMPLES

Security considerations sections for most individual functions can follow this simple formula:

1. Provide one or two sentences describing each potential security problem.
2. Provide one or two sentences describing how to avoid each potential security problem.
3. Provide a short example in code.

This is an example security considerations section for the strcpy(3) manual page:

The **strcpy**() 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 **strcpy**(). Instead, use **strncpy**() and ensure that no more characters are copied to the destination buffer than it can hold. Do not forget to NUL-terminate the destination buffer, as **strncpy**() will not terminate the destination string if it is truncated.

Note that **strncpy**() 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 strcpy is bad behavior.  Do not use strcpy()!
         */
        (void)strcpy(onstack, arbitrary_string);     /* BAD! */
#elif defined(BETTER)
```

```
        /*
         * The following two lines demonstrate better use of
         * strncpy().
         */
        (void)strncpy(onstack, arbitrary_string, sizeof(onstack) - 1);
        onstack[sizeof(onstack - 1)] = '\0';
#elif defined(BEST)
        /*
         * These lines are even more robust due to testing for
         * truncation.
         */
        if (strlen(arbitrary_string) + 1 > sizeof(onstack))
                err(1, "onstack would be truncated");
        (void)strncpy(onstack, arbitrary_string, sizeof(onstack));
#endif
}
```

Security considerations sections for tools and commands are apt to be less formulaic. Let your list of potentially-violated security requirements be your guide; explain each one and list a solution in as concise a manner as possible.

This is an example security considerations section for the rtld(1) manual page:

Using the LD_LIBRARY_PATH and LD_PRELOAD environment variables, malicious users can cause the dynamic linker to link shared libraries of their own devising into the address space of processes running non-set-user-ID/group-ID programs. These shared libraries can arbitrarily change the functionality of the program by replacing calls to standard library functions with calls to their own. Although this feature is disabled for set-user-ID and set-group-ID programs, it can still be used to create Trojan horses in other programs.

All users should be aware that the correct operation of non set-user-ID/group-ID dynamically-linked programs depends on the proper configuration of these environment variables, and take care to avoid actions that might set them to values which would cause the run-time linker to link in shared libraries of unknown pedigree.

SEE ALSO

groff_mdoc(7), security(7), sprog(7)

Edward Amoroso, AT&T Bell Laboratories, *Fundamentals of Computer Security Technology, P T R Prentice Hall*, 1994.

Ken Thompson, "Reflections on Trusting Trust", *Association for Computing Machinery, Inc.*, *Communications of the ACM*, Vol. 27, No. 8, 761-763, August, 1984.

HISTORY

The **sdoc** manual page first appeared in FreeBSD 5.0.

AUTHORS

Tim Fraser <*tfraser@tislabs.com*>, NAI Labs CBOSS project
Brian Feldman <*bfeldman@tislabs.com*>, NAI Labs CBOSS project