

LEGIBILITY NOTICE

A major purpose of the Technical Information Center is to provide the broadest dissemination possible of information contained in DOE's Research and Development Reports to business, industry, the academic community, and federal, state and local governments.

Although a small portion of this report is not reproducible, it is being made available to expedite the availability of information on the research discussed herein.

Received by OSTI

APR 05 1989

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract number W-7408-ENG-88

LA-UR--89-861

DE89 009390

TITLE HOW TO WRITE APPLICATION CODE EVEN A SECURITY AUDITOR COULD LOVE

AUTHOR(S) Gail L. Barlich

SUBMITTED TO 12th Computer Security Group Conference, Amarillo, Texas, May 2-4, 1989

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, method, or process disclosed, or represents that its use would not infringe privately owned rights. References herein to any specific commercial product, process, or service by trade name, trademark, name of manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

MASTER



This report is the property of Los Alamos National Laboratory. It is loaned to you; it and its contents are not to be distributed outside your organization.

This report is the property of Los Alamos National Laboratory. It is loaned to you; it and its contents are not to be distributed outside your organization.

Los Alamos National Laboratory Los Alamos, New Mexico 87545

HOW TO WRITE APPLICATION CODE EVEN A SECURITY
AUDITOR COULD LOVE

Gail Barlich
Safeguard Systems
Box 1663, Mail Stop E541
Los Alamos National Laboratory
Los Alamos, NM 87545
505) 667-7777

In the past the application programmer was frequently isolated from the computer security professional. The target machine might have various access controls and security plans, but when the programmer delivered a new application, it was rarely scrutinized from a security standpoint. Security reviews of application code are now being used to overcome this apparent oversight, but these reviews are often hampered by a lack of knowledge among programmers of techniques that make code secure and facilitate security analysis of the code. This paper informally describes fifteen general principles for producing good code that is easily reviewed. This paper is not a formal guideline, but is intended as an inside view of how one reviewer looks at code from a security standpoint.

1. I distrust any code that uses obscure features in the language and/or avoids efficient features consistently. The obscure features tend to be the ones hackers use, the efficient ones are usually well-debugged and safe. I wonder if the programmer is sloppy, trying to be flashy, or hiding problems.
2. Any code with incomplete or incorrect internal documentation is suspect. When I see such inconsistency, I have to question whether the programmer understood what the code does or if the code was rushed into incorrectness. Countless security implications are possible if the programmer really hasn't thought things out. I personally use a standard header with the name of the routine, what it does, how it is called, what it calls, inputs, outputs, and a short modification history.

7. When it appears that security features were implemented without orderly planning, I have to believe that the code cannot be trusted. If five similar but different routines are used for password collection, I doubt that all five accomplish the task with the same level of skill. Probably at least one of them will be untrustworthy. Yet if I see just one password routine called in every place where a password is required, I can study the routine and know that every call behaves the same.
8. Large systems in secure environments must have some kind of logging built into them. The log not only catches abuse of the system after delivery, but it can be used to confirm correctness also. I question whether the programmer really checked the step-by-step behavior of the system if no log was built in. There is also a delicate balance between keeping too much information and keeping too little.
9. I always ask about account management and where the code will go in the machine. There are countless concerns about account privileges and protection of files. Ideally, the executable code should go into a limited captive account. Data files and source code should be in accounts that cannot be reached if the captive account is compromised. I realize that not every system can be configured in this way, but the goal of protecting code and data files from casual modification or study must be considered.
10. Plan for security even if the customer seems bored. If they refuse to accept security features now, leave room for them anyway. More security requirements for application codes are inevitable. Adding security features to existing code is difficult and may force undesirable compromises if poor planning was done when the code was designed.
11. Plan for expansion of the code. During design, it pays to list possible changes that could occur and consider how to plan for them. A design with such foresight will survive a major upgrade and still be easy to study and certify. I shudder when I must

examine a code with messy additions that could have been foreseen; frequently the addition breeds undesired side-effects on the original code.

12. I firmly believe that team projects must have constant peer review of the design and code for a viable system to emerge. Running peer reviews (known as "walkthroughs") is difficult, but publications are available with procedures and forms that minimize personnel problems. The walkthrough ensures consistency and correctness throughout the project. Such checking must be done during a security review, so it makes sense to do it during development when problems can be easily corrected. The programmers have the responsibility to produce good code; the security reviewer should merely check that the programmer did his or her job.
13. Where a username/password protection scheme is used I always examine the choices made for such protection with a very critical eye. Why have passwords at all if the password file is easily examined or if passwords appear on the screen as they are typed? A password system must be very robust or it provides no security at all.
14. I distrust large systems with a variety of coding styles represented. During review I must shift gears each time a new style appears and remember each programmer's weaknesses. Using a style guide makes for clear code that is easy to examine as a whole. Most languages have a variety of style guides available, or there may be in-house versions available from other organizations. Using a style guide with peer review can produce code that is unified and easy to read.
15. Security reviewers groan when most of the routines in a system are over 50 lines. Algorithms are easier to understand if they are coded into small, conceptually unified pieces. Good planning and design before coding should make such partitioning possible.

In summary, writing code to facilitate security analysis is really a matter of producing professional, quality code. Clear, well-documented code is usually secure. A thorough reading will confirm that it does its job well and probably won't admit surprises. Analysis of poorly written code with little documentation is tedious and unsure. Hidden surprises are more likely and the reviewer is forced to guess at the thoughts of the programmer. It is my opinion maverick programmers who refuse to craft neat, well-documented systems should not be allowed to work on code for secure applications. The maintenance and analysis of such systems is just too tenuous for all professionals involved.