Secure Coding Standards

James W. Moore
The MITRE Corporation

Robert C. Seacord
Software Engineering Institute

Inherent weaknesses in programming languages contribute to software vulnerabilities. Increasingly, organizations are producing standards to improve software security. Current efforts to develop software security standards are surveyed, and two such efforts are described in detail. An international standards group is writing a document providing guidance to users of programming languages on how to avoid the vulnerabilities that exist in the programming language selected for a particular project. Carnegie Mellon University's (CMU's) Computer Emergency Response Team (CERT) is developing secure coding practices for the C and C++ programming languages[1].

Today's dependency on networked software systems has been matched by an increase in the number of attacks against governments, corporations, educational institutions, and individuals. These attacks result in the loss and compromise of sensitive data, system damage, lost productivity, and financial loss. To address this growing threat, the introduction of software vulnerabilities during development and ongoing maintenance must be significantly reduced, if not eliminated.

It is no secret that common, everyday software defects cause the majority of software vulnerabilities. Poor development practices cause numerous delivered defects, some of which can lead to vulnerabilities. Software developers repair vulnerabilities as they are reported; cycles of patch and install follow. However, there are so many patches to install that system administrators cannot keep up with the job. Often the attackers analyze the patches for clues to attacking unpatched systems. Sometimes the patches themselves contain security defects. The patch-oriented strategy of responding to security defects is not working. There is need for a prevention and early security defect-removal strategy.

Software Vulnerabilities

One example of a relatively common programming error that can lead to software vulnerabilities involves the use of formatted output functions in C and C++, as well as some other common languages. For example, a malicious user is able to manipulate the string variable named *output* in the following statement:

```
printf(output);
```

The malicious user will be able to exploit this vulnerability to execute code with the permissions of the vulnerable process [1]. This is largely a consequence of the little known %**n** conversion specifier that instructs the formatted output function to write an integer value to a specified address. The error is the passing of untrusted data as a format string. The solution is as easy as providing a static format specification:

```
printf("%s", output);
```

Or, even more succinctly, using the puts() function so that no formatting is necessary or possible, as shown in the following example:

```
puts(output);
```

Format output functions can also lead to vulnerabilities in other languages as is shown in the following examples:

- **Perl:** could alter values, the consequence of which depends on how the value is used.
- **PHP:** log avoidance (fails quietly).
- **Python:** information leak or denial-of-service (DoS) attacks.
- **Ruby:** DoS attacks.

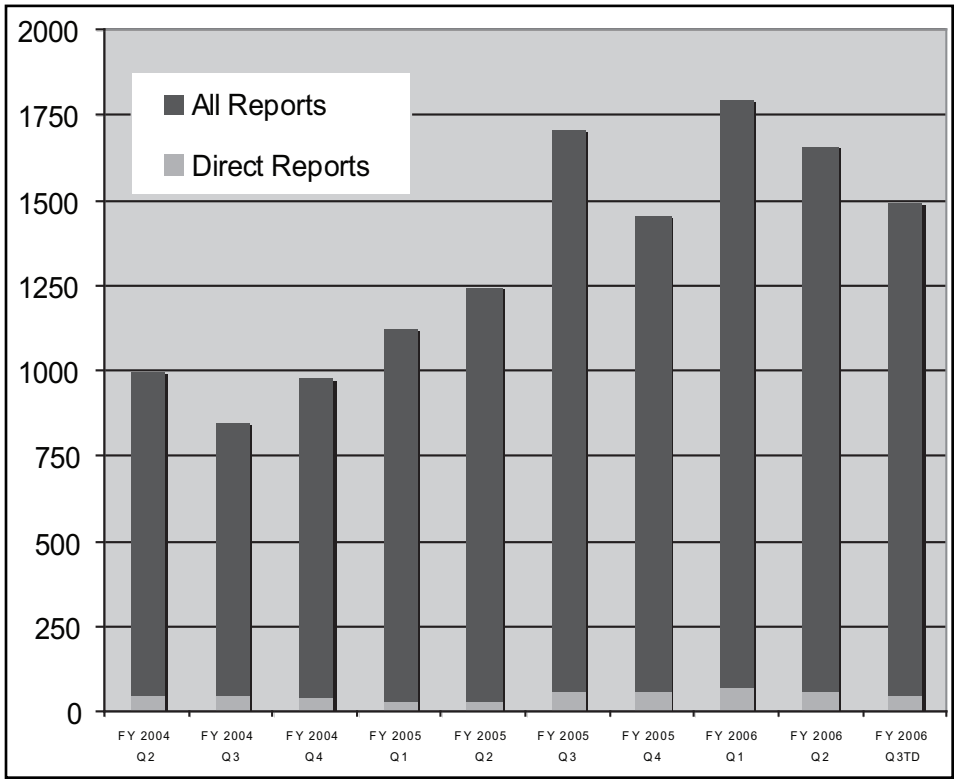
As long as developers are unaware of the security risks, it is likely that common programming errors, such as allowing untrusted data to be incorporated into a format string, will result in software vulnerabilities being operationally deployed.

If root causes of software vulnerabilities are not addressed, software vulnerability reports are likely to continue on the upward trend as shown in Figure 1. Nearly 4,000 vulnerabilities were reported in the first half of 2006 alone.

International Standardization Efforts

Inherent weaknesses in programming languages such as C, C++, Fortran, Ada, and COBOL are a contributing factor to software vulnerabilities. Many of these languages were specified in the days when ubiquitous networking language designers

Figure 1: *Vulnerabilities Reported to CERT/Coordination Center (CC)*



were not greatly concerned with the prospect of attacks by external parties. Many languages were predominately designed for flexibility, ease of use, and performance. As a result, many of the languages, in effect, invite coders to use insecure coding constructs. One result has been the growth of usage guidelines that inform and encourage coders to use secure alternatives to easily attacked constructs. Another result is the development of recent projects by standards bodies to provide secure alternatives.

Increasingly, standards organizations are working on ways to improve software security. Accomplishing change through standards organizations can be harder than accomplishing change at any other organizational level, but when successful, can have a broader impact across the industry. The international standards bodies – International Organization for Standardization (ISO) and International Electrotechnical Commission (IEC) – are working on a number of projects that affect software security:

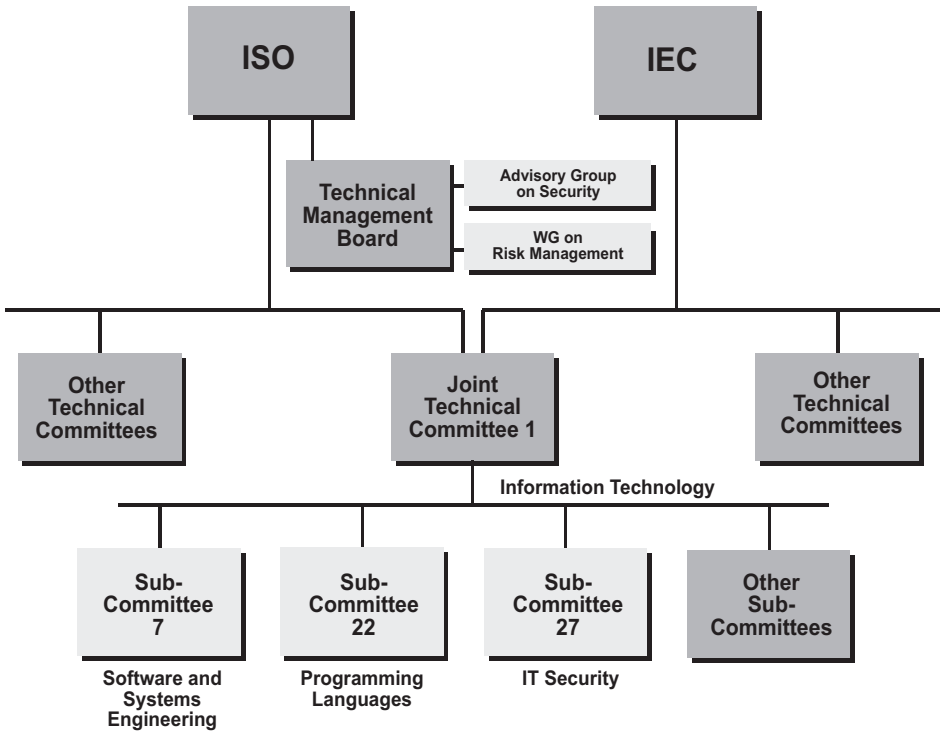
- The ISO Technical Management Board (TMB) performs strategic planning and coordination for ISO. Currently, its advisory group on security is coordinating standards efforts. The ISO TMB working group on risk management is providing overall guidance for risk management.
- ISO/IEC Joint Technical Committee (JTC) 1/Standards Committee (SC) 27 is responsible for computing security. Many of its projects concern methods for protecting deployed software <www.ni.din.de/sixcms/detail.php?id=10172>.
- ISO/IEC JTC 1/SC 7 has the responsibility for systems and software engineering. Its standards provide a baseline for the responsible practice of systems and software engineering, including software assurance <www.jtc1-sc7.org/>.
- ISO/IEC JTC 1/SC 22 has the responsibility for programming languages, including their effect on improved software security <www.open-std.org/jtc1/sc22/>. For example, SC 22/ Working Group (WG) 14, the group that is responsible for the standardization of the C programming language, is developing a technical report on C library functions that incorporate bounds checking to mitigate against buffer overflow vulnerabilities [2].

Figure 2 illustrates the relationship of the relevant standards committees.

Other Working Group Vulnerabilities (OWGV)

All programming languages have constructs that are imperfectly defined, implementation-dependent, or difficult to use correctly. As a result, software programs can execute differently than intended by the writer. In some cases, these vulnerabilities can be exploited by an attacker to compromise the safety, security, and privacy of a system.

Figure 2: *Relevant International Standard Committees*



A new SC 22 group, the OWGV, is addressing the issue of programming language vulnerabilities. The goal of the OWGV <http://aitc.aitcnet.org/isai/>, as described in the proposal for a new work item, is the following:

> … to prepare comparative guidance spanning a large number of programming languages, so that application developers will be better informed regarding the vulnerabilities inherent to candidate languages and the costs of avoiding such vulnerabilities. An additional benefit is that developers will be better prepared to select tooling to assist in the evaluation and avoidance of vulnerabilities. [3]

ISO/IEC JTC 1/SC 22/OWGV is writing a document containing guidance to users of programming languages on how to avoid the vulnerabilities that exist in the programming language selected for a particular project. Implicitly, the guidance may also be helpful in selecting a language for a particular project. Finally, the report may be helpful in choosing tools to assist in evaluating and avoiding vulnerabilities. The tentative schedule calls for publishing this document in January of 2009.

Although work has just begun, it is anticipated that the group's technical report will provide comparative guidance covering a wide variety of languages. Readers would be able to study the vulnerabilities of particular languages they already know well; in addition, readers would be able to apply their existing knowledge of vulnerabilities in the context of programming languages that are unfamiliar.

Currently, the group enjoys the participation of representatives from many of the important programming languages and hopes to attract more. The group plans to obtain information about vulnerabilities and their treatment from initiatives like the common vulnerabilities and exposures database <http://cve.mitre.org> and the CERT secure coding initiative.

CERT Secure Coding Initiative

Additional efforts in developing secure coding standards are originating outside of formal standards organizations. CMU's Software Engineering Institute (SEI) CERT program has deployed a secure coding Web site at <www.securecoding.cert.org> to cooperate with the software development community in codifying a

practical and effective set of secure coding practices for popular programming languages. These coding practices can be used by software developers to eliminate vulnerabilities before software is operationally deployed.

CERT's initial efforts are focused on the development of secure coding practices for the C and C++ programming languages. C and C++ were selected because a large percentage of critical infrastructure is developed and maintained using these programming languages. C and C++ are popular and viable languages, although they have characteristics that make them prone to security flaws. The CERT C programming language secure coding standard is scheduled for publication in January 2008 while the C++ standard is not scheduled for publication until January 2009. However, working drafts for both documents are available on the secure coding Web site.

There are numerous available sources, both online and in print, containing coding guidelines, best practices, suggestions, tips, and industry-specific standards such as the Motor Industry Software Reliability Association (MISRA) Guidelines for the use of the C language in critical systems [4]. However, none of these sources provide a prescriptive set of secure coding standards that can be uniformly applied in the development of a software system. This conclusion is reinforced by the Secure Software Assurance Common Body of Knowledge [5], published by the DHS, which laments the *lack of public standards as such for secure programming.*

The secure coding practices proposed by CERT are based on standard language specifications as defined by official standards organizations (such as ISO/IEC) or by *de facto* standard language specifications. CERT is not an internationally recognized standards body, but it is working with organizations such as ISO/IEC to advance the state of the practice in secure coding. The ISO/IEC JTC1/SC22 WG14 international standardization working group for the programming language C, for example, has offered to provide direction in the development of the C language secure coding practices and to review and comment on drafts of the informal CERT standard.

The goal of the CERT work is to encourage organizations to develop their own coding standards to be applied on all of their projects. The organizational coding standard would codify a set of rules that are necessary (but not sufficient) to ensure the security of software systems developed in the respective programming languages [6].

A secure coding standard consists of *rules* and *recommendations*. Coding practices are defined to be rules when all of the following conditions are met:

1. Violation of the coding practice will result in a security flaw that may result in an exploitable vulnerability.
2. An enumerable set of exceptional conditions (or no such conditions) in which violating the coding practice is necessary to ensure the correct behavior for the program. One example of a rule with an exception condition is to *ensure that integer operations do not result in an overflow*. While overflow generally indicates an error condition, if the code was designed assuming module behavior then it is necessary to provide an exception for overflows resulting from this behavior.
3. Conformance to the coding practice can be verified.

"The goal of the CERT work is to encourage organizations to develop their own coding standards to be applied on all of their projects. The organizational coding standard would ... ensure the security of software systems developed in the respective programming languages."

Rules must be followed to claim compliance with a standard unless an exceptional condition exists. If an exceptional condition is claimed, the exception must correspond to a predefined exceptional condition and the application of this exception must be documented in the source code.

Recommendations are guidelines or suggestions. Coding practices are defined to be recommendations when all of the following conditions are met:

1. Application of the coding practice is likely to improve system security.
2. One or more of the requirements necessary for a coding practice to be considered a rule cannot be met.

Relationships Between Efforts

CERT representatives participating in the ISO/IEC working group on guidance for avoiding vulnerabilities through language use are coordinating their efforts. While the ISO/IEC group is working on providing language-independent guidance, the CERT effort is working on developing and consolidating the language-specific guidance that provides the foundations for the ambitious goals of OWGV.

CERT's efforts in identifying and documenting secure coding practices for C and C++ will contribute to the standardization of these practices and advance the goals of the OWGV, while the OWGV effort provides a framework for CERT language-specific efforts.

Summary

Efforts have now begun to codify secure coding practices both at the international and organizational levels. The success of the secure coding standards depends on the active involvement of members of the software development communities. To become involved in the OWGV group, visit <www.aitcnet.org/isai/> or contact the convener. To contribute to the CERT secure coding standards, go to <www.securecoding.cert.org> and review or comment on the existing content or submit new ideas for secure coding practices.◆

Acknowledgements

Thanks to Hal Burch for his information on format string vulnerabilities in languages other than C and C++ and to John Benito for his assistance in developing this article. Thanks also to reviewers Pamela Curtis, Chad Dougherty, and Fred Long.

References

1. Seacord, Robert C. Secure Coding in C and C++. Boston, MA: Addison-Wesley, 2005 <www.cert.org/books/secure-coding>.
2. ISO/IEC JTC1 SC22 WG14. Information Technology – Programming Languages, Their Environments and System Software Interfaces – Extensions to the C Library, – Part I: Bounds- Checking Interfaces. ISO/IEC TR 24731. Geneva, Switzerland: ISO, 2006 <www.open-std.orgjtc1/sc22/wg14/www/docs/n1146.pdf>.
3. ISO/IEC JTC 1/SC 22. New Work Item Proposal for Guidance to Avoiding Vulnerabilities in Programming Languages Through Language Selection and Use. 2005 <www.open-std.org/jtc1/sc22/open/n3913.htm>.
4. MIRA Limited. MISRA C: 2004 Guidelines for the Use of the C Lan-

guage in Critical Systems. Warwickshire, UK: MIRA Limited, 2004.
5. Redwine, Jr. Samuel T., Ed. Software Assurance: A Guide to the Common Body of Knowledge to Produce, Acquire, and Sustain Secure Software. Draft version 1.05. Aug. 2006.
6. Seacord, Robert C. "Secure Coding Standards." Static Analysis Summit. NIST Special Publication 500-262. Gaithersburg, MD: NIST, 2006. 14-16 <http://samate.nist.gov/docs/NIST_Special_Publication_500-262.pdf>.

Note

1. The nomenclature for international standards groups represents a hierarchical organization. The international standards committee for information technology is a JTC of two international standards-makers, the ISO and the IEC, and is therefore called ISO/IEC JTC 1. It subdivides its work among numbered subcommittees: SC 7 deals with software and systems engineering, SC 22 with programming languages, and SC 27 with computing security. SCs subdivide their work among numbered or lettered WG and OWGs.

About the Authors

James W. Moore is a 35-year veteran of software engineering in IBM and, now, the MITRE Corporation. He is an executive editor of the IEEE Computer Society's *Guide to the Software Engineering Body of Knowledge* and a member of the editorial board of the *Encyclopedia of Software Engineering*. He participates in international standardization related to software and systems engineering as well as programming languages. Moore is a fellow of the IEEE and a charter member of the IEEE Computer Society's Golden Core.

MITRE Corporation
7515 Colshire DR
H505
McLean, VA 22102-7508
Phone: (703) 983-7396
Fax: (703) 983-1279
E-mail: moorej@mitre.org

Robert C. Seacord is a senior vulnerability analyst at the CERT/CC at the SEI. He is the author of *Secure Coding in C and C++* and co-author of *Building Systems from Commercial Components* and *Modernizing Legacy Systems*, as well as more than 50 papers on software security, component-based software engineering, Web-based system design, legacy-system modernization, component repositories and search engines, and user interface design and development. Seacord also has worked at the X Consortium where he developed and maintained code for the Common Desktop Environment and the X Window System.

SEI
Pittsburgh, PA 15213
Phone: (412) 268-7608
Fax: (412) 268-6989
E-mail: rcs@cert.org

Web Sites

BuildSecurityIn

https://buildsecurityin.us-cert.gov/daisy/bsi/home.html

As part of the Software Assurance program, Build Security In (BSI) is a project of the Strategic Initiatives Branch of the National Cyber Security Division (NCSD) of the Department of Homeland Security. The Software Engineering Institute (SEI) was engaged by the NCSD to provide support in the Process and Technology focus areas of this initiative. The SEI team and other contributors develop and collect software assurance and software security information that helps software developers, architects, and security practitioners to create secure systems. BSI content is based on the principle that software security is fundamentally a software engineering problem and must be addressed in a systematic way throughout the software development life cycle. BSI contains and links to a broad range of best practices, tools, guidelines, rules, principles, and other knowledge that can be used to build security into software in every phase of its development.

National Institute of Standards and Technology: Computer Security Resource Center

www.csrc.nist.gov

The Computer Security Division – (893) is one of eight divisions within National Institute of Standards and Technology's (NIST) Information Technology Laboratory. The mission of NIST's Computer Security Division is to improve information systems security by: raising awareness of information technology (IT) risks, vulnerabilities and protection requirements, particularly for new and emerging technologies; researching, studying, and advising agencies of IT vulnerabilities and devising techniques for the cost-effective security and privacy of sensitive federal systems; developing standards, metrics, tests and validation programs; and developing guidance to increase secure IT planning, implementation, management and operation.

Computer Emergency Readiness Team Coordination Center

www.cert.org

The Computer Emergency Readiness Team Coordination Center is a center of Internet security expertise, located at the Software Engineering Institute (a federally funded research and development center operated by Carnegie Mellon University). The team studies Internet security vulnerabilities, researches long-term changes in networked systems, and develops information and training to help you improve security.

Committee on National Security Systems

www.cnss.gov

Under Executive Order 13231 of October 16, 2001, Critical Infrastructure Protection in the Information Age, the President designated the National Security Telecommunications and Information Systems Security Committee as the Committee on National Security Systems (CNSS). The CNSS provides a forum for the discussion of policy issues, sets national policy, and promulgates direction, operational procedures, and guidance for the security of national security systems.