ISO/IEC JTC 1/SC 22/WG 14/N 1577

Rationale for a C Secure Coding Analysis Technical Specification

David Keaton, CERT

2011-09-20

In recent years, organizations that develop software in C have pursued mechanical assistance for secure coding with increased urgency. This is often achieved with source code security scanners, in the form of static analysis tools. (Static analysis simply means that the program is not actually run. The tool examines the source code.)

Although this subject area is still under development, it has already become clear that the community could benefit from some standards-related support. To that end, WG 14 created the C Secure Coding Rules Study Group (CSCR SG) two years ago. Its purpose is to study the problem of creating secure coding rules for C, and recommend a course of action.

The CSCR SG is now ready to make its recommendation. At the October, 2011, meeting of WG 14, the study group plans to recommend creating an ISO Technical Specification on secure coding. The study group will also introduce a document it has developed for use as a potential base document for that Technical Specification. The goal is to specify a minimum level of capabilities for source code analyzers, rather than specifying what actions a programmer must take.

For such an undertaking to be worthwhile, there must be a need for standardization, a market that is underserved by standards, and a market that is large enough. In addition, for a Technical Specification to be appropriate, it must fit the situation better than other standards-related alternatives.

Need for Standardization

The application of static analysis to security has evolved in an ad hoc manner. This is useful from the point of view of exploring a new market to see what works. However, it has resulted in a fragmented market, with different vendors addressing different security issues, and no way for a purchaser to specify the minimum requirements of a static analysis tool. Now that the shape of security needs is becoming clearer, there is a need for a specification that says "For an analysis tool to conform to this specification, it must be able to do at least *this much*," where *this much* is spelled out in detail.

By imposing a floor on analysis capabilities rather than circumscribing them completely, a specification can allow for continued improvements while still giving customers a way to know what they are buying.

Underserved Market

The largest underserved market in security is ordinary, non-security-critical code. For the purposes of the present work, the CSCR SG has considered the security-critical nature of code to depend on its purpose rather than its environment.

The UNIX finger daemon (`fingerd`) is an example of ordinary code, even though it may be deployed in a hostile environment. A user runs the client program, `finger`, which sends a user name to `fingerd` over the network, which then sends a reply indicating whether the user is logged in, and a few other pieces of information. The function of `fingerd` has nothing to do with security. However, in the late 1980s, Robert Morris compromised `fingerd` by triggering a buffer overflow, allowing him to execute arbitrary code on the target machine. The Morris worm could have been prevented from using `fingerd` as an attack vector by preventing buffer overflows, regardless of whether `fingerd` contained other types of bugs. (In fact, the Morris worm was the exact reason that CERT was created.)

By contrast, the function of `/bin/login` is purely related to security. A bug of any kind in `/bin/login` has the potential to allow access where it was not intended. This is security-critical code.

Similarly, in safety-critical code, such as software that runs an X-ray machine, any bug at all could have serious consequences. In practice, then, security-critical and safety-critical code have the same requirements.

There are already standards that address safety-critical code, and therefore security-critical code. The problem is that because they must focus on preventing essentially all bugs, they are required to be so strict that most people outside the safety-critical community do not want to use them. This leaves ordinary code like `fingerd` unprotected.

The CSCR SG proposes to address this underserved market with a Technical Specification.

Large Enough Market

Static analysis tools are available from a growing, but still small number of vendors. However, increasingly, static analysis techniques are being incorporated into compilers. This happens because much of the infrastructure for static analysis is already present in a compiler, so the incremental effort to add security scanning checks is smaller than would otherwise be the case.

Because the proposed C secure coding rules constrain only analyzers, and not compilers in general, they do not impose a burden. Because many compilers are capable of being upgraded to become static analysis tools as well, though, the secure coding rules provide guidance and help for a much larger market than just the static analyzers that are available today.

Technical Specification

Given the need for standardization discussed above, it would normally be logical to propose a standard as the solution. However, because this subject is still under development, we have a situation where there is a need, and yet a full standard is premature.

Another publication option is a Technical Report. However, the ISO/IEC JTC 1 directives reserve this option for "data of a different kind from that which is normally published as an International Standard." Furthermore, Technical Reports are not to contain any normative language. The study group believes that only a normative specification can fulfill the needs of the community.

A Technical Specification is the best middle ground. It gives WG 14 the right, but not the obligation, to make the document into an International Standard at a later date. In a fluid subject area such as source code security scanners currently, this is the most appropriate solution.

Conclusion

The C Secure Coding Rules Study Group is proposing what it believes is the right solution for today, with room for further developments in the future. It addresses the need for static analyzer customers to know that the tool they purchase does "at least *this much*" toward detecting security problems in their code. It addresses the underserved market of ordinary code, rather than being appropriate only for safety- or security-critical code. It can help, but does not require, compiler vendors to enter the static analysis market, thus addressing a much larger potential market than just today's static analysis tools. Finally, a Technical Specification strikes the right balance between the authority of an International Standard and the flexibility of a Technical Report.