Software Assurance During Maintenance

Paul E. Black
U.S. National Institute of Standards and Technology (NIST)
paul.black@nist.gov

Abstract

Software development, testing, and maintenance tools must yield assurance information in a standardized form. Developers can use this information to argue that the software is adequate for its use and secure enough for the risk.

NIST's Software Assurance Metrics And Tool Evaluation (SAMATE) project is developing specifications for software assurance tools. These specifications can include optional features for assurance information reports, encouraging tools to provide them. During maintenance, developers can collect this information to make explicit assurance cases.

1. Introduction

The old days of software engineering for a benign environment are gone. Today most applications are exposed to the Internet. Even internal, scientific, or control applications may have far-reaching effects: consider Ariane 5. Merely running regression tests after a software change is becoming less and less adequate.

In the traditional model the assurance case was implicit: if it passes code reviews and regression tests, it is good enough for use. Today the software engineer must consider what arguments make a convincing case that the software to be released is sufficiently dependable. The engineer must assemble evidence from many different sources.

2. The SAMATE Project

The U.S. National Institute of Standards and Technology (NIST) began the Software Assurance Metrics And Tool Evaluation (SAMATE) project in June 2004. Funded in part by the U.S. Department of Homeland Security, the project examines software development and testing methods to improve software assurance throughout the software development life cycle. We are surveying software assurance tools and techniques to identify deficiencies and lead the development of a research plan to address them. In the future we will lead a set of studies and experiments to measure the effectiveness of tools and techniques, that is, their ability to produce secure software. We do *not* want to duplicate existing datasets, surveys, or work. Many are working to assess tools, for example, [7] and [1]. Rather, we want to contribute to existing work, refer to these resources, and complement them.

Why is NIST interested? NIST is a non-regulatory agency and has over a century of experience in standards and measurement. It serves as a neutral party and as a long-term repository for standard reference materials, datasets, and other items. Also, NIST already has accreditation activities, such as the National Voluntary Laboratory Accreditation Program (NVLAP) and the National Information Assurance Partnership (NIAP).

As a first step, we are developing specifications for classes of software assurance tools, such as source code analyzers and web application scanners. The specifications are precise enough for us to also develop test plans and other test material, including a reference dataset of good code and code with known vulnerabilities.

In addition to the specific test material, thousands of research and example cases are publicly available in the SAMATE Reference Dataset (SRD) [3]. There are tiny test cases for specific weaknesses, large cases from actual code, synthetic code from research projects, and code donated by vendors. The SRD is growing and will eventually cover all classes of software security vulnerabilities with versions for different languages and different platforms or environments, where applicable. The SRD will have examples for all phases of the software development life cycle from use cases to requirements to executables. We welcome and actively seek contributions from vendors, researchers, and users.

These specifications and tests can help assure users that the tools are effective. We include requirements for optional features to guide vendors to improve their products. One optional feature would be assurance evidence.

3. Explicit Evidence From Tools

All tools used in the software life cycle, from compilers to IDEs, from linkers to test generators, should yield explicit assurance information. Test coverage metrics or digital certificates from source code analyzers are basic examples.

To be most useful, this information should be in an easy-to-process format, such as XML, and follow industry-wide conventions. Although there is no widely-accepted format for security assurance evidence, such as vulnerabilities and their severity, there are several efforts in such a direction.

Djenana Campara and James D. Baker are co-chairs of the Software Assurance Special Interest Group (SwA SIG) formed in February 2006. The goal is to "to establish a common framework for analysis and exchange of information related to software trustworthiness." [4] With the framework, design claims and arguments, assurance evidence produced by tools, and security requirements from existing standards could be combined and analyzed.

Jeff Williams maintains that software should have a "security facts" label [6], in the spirit of food labels or material safety data sheets. Daniel J. Quinlan proposed that analyzer certificates or evidence could be attached to the executable [2], as in proof-carrying code, see for instance [5].

With a widely-accepted convention for assurance evidence, the SAMATE project can include requirements for optional features in their specifications for software assurance tools. With clear specifications and adequate tests, researchers and vendors would have the incentive to include such assurance information reports in their tools. During maintenance this information could be accumulated.

4. Making the Assurance Case

Not surprisingly, a software assurance case potentially gets evidence from many different sources, such as

- formal correctness proofs of algorithms or protocols,
- code reviews,
- compiler safety enforcement, e.g., add bounds checks,
- code scanners to look for security vulnerabilities,
- testing with different coverage and criteria, and
- live testing with "alpha" or "beta" users.

A grossly simplified assurance case might be: if test suite branch coverage exceeds 90% for all modules and no errors are exposed by the test suite, the software is acceptable. Depending on the risk, a software assurance case may be simple and informal or more complex and quite formal.

When software is first conceived and written, creating an extensive assurance case, with assumptions, arguments, evidence and other elements given explicitly, may be an acceptable part of developing test plans and acceptance criteria. However, nobody can afford complete manual reanalysis after every incremental change. An explicit assurance case, aided by appropriate tools, can be quickly reviewed during maintenance. On the other hand, with new attacks being invented every day, simply reusing the initial assurance case over and over is unacceptable. Maintainers must be prepared to reconsider the assurance case.

Small changes are likely to have small effects on the assurance case, which can be adapted quickly. Perhaps just a few more test cases are needed. Maybe rerunning a source code analyzer, updated with the latest weakness signatures, suffices. However, reviewing the assurance case may show that a large change requires a whole new class of assurance activity. For instance, adding a web interface to an application may require penetration testing before release.

5. Summary

Never before has so much of the world depended on vastly spread software-run systems. Never before has there been such a risk of computer malfeasance, from spyware to identity theft to electronic warfare. Never before has the decay of morals and the lowering of societal norms combined with global communication to allow the few to harm the many in so many ways. Our professional ethics demand that we support and encourage secure software.

Tools already used in software development and maintenance can automatically supply information for assurance cases. This information can be gathered to make a case that software is effective and secure enough for its intended use.

References

[1] F. Abbott and J. Saur. Comparison of code checker technologies for software vulnerability evaluation. Technical report, Joint Systems Integration Command, April 2005.

[2] D. J. Quinlan. Lawrence Livermore National Laboratory. personal communication, Mar. 2006.

[3] SAMATE reference dataset. Available from http://samate.nist.gov/SRD (accessed June 2006).

[4] Software Assurance SIG. Highlights of technical meeting, Tampa, FL. Available from http://swa.omg.org/swa_info.htm (accessed June 2006), Feb. 2006.

[5] M. Wildmoser. *Verifi ed Proof Carrying Code*. PhD thesis, Institut für Informatik, Technische Universität München, 2005.

[6] J. Williams. Unsafe at any (CPU) speed: Why we make the same security mistakes over and over again. Available from http://www.aspectsecurity.com/documents/Aspect_HCSS_Brief.ppt (accessed June 2006), Mar. 2005.

[7] M. Zitser, R. P. Lippmann, and T. Leek. Testing static analysis tools using exploitable buffer overflows from open source code. In *Proc. 12th Internt'l Symp. on Foundations of Software Engineering*, pages 97–106. ACM SIGSOFT, 2004.