

The Verified Software Roadmap: Experiments

N. Shankar

shankar@csl.sri.com

URL: <http://www.csl.sri.com/~shankar/>

Computer Science Laboratory
SRI International
Menlo Park, CA

Formalization is an experimental science. Dana Scott

Overview

Key question: *What verification experiments can be conducted to demonstrate the feasibility and value of industrial-scale verification?*

We outline a number of possible projects where formal specification and the development of verified software can have a significant impact.

A vast collection of open source software is available from resources like SourceForge, but we need to choose carefully to optimize resources and maximize impact.

Outline

- Libraries: OpenSSL, Eiffel, Java, scientific software, computer algebra.
- Embedded devices: Device drivers, Javacard, medical devices, lightweight runtime environments.
- Systems: Hypervisor, operating systems security, TCP/IP, middleware platforms, web server, web browser.
- Applications: Avionics, automotive systems, E-voting, SCADA.

Libraries

Libraries

Software implementing a family of functions through a given application programmer interface (API) for online or offline use.

The chosen libraries must be heavily used in critical applications in order to justify the investment in verification.

The verification can range from requirements modeling to the absence of runtime errors and complete verification.

Formally specifying/deriving the interfaces for library operations is a key challenge.

We expect some of the initiative for specification and verification to be driven by the developers.

OpenSSL

OpenSSL is an open source library for various cryptographic operations.

It implements various standards for encryption and certification including [ASN.1](#), [RSA](#), [X.509](#), [passwd](#), [pkcs12](#), [S/MIME](#), [MD5](#), and [SHA1](#).

The verification can validate the mathematical properties of these protocols such as invertibility, authentication, correspondence, and malleability.

The API to synthesize attacks (test cases) and verify cryptographic protocols.

Eiffel/Java Libraries

The object-oriented programming language Eiffel includes libraries for data structures, window systems, graphics, networking, data storage and access, and web site development.

Eiffel employs a *Design-by-Contract* methodology where methods have preconditions, post-conditions, ensures, and invariant statements which can be tested as assertions.

The Eiffel libraries yield access to a large body of well-annotated code.

Java also has open source libraries: JFC/Swing, STL, IDEs, numerics, database connectivity, networking, web services, and XML.

Verification Tool Libraries

Verification itself relies quite heavily on libraries for Boolean simplification (BDDs), polyhedral manipulation, linear programming, term manipulation, rewriting, and computer algebra.

Verifying these libraries is a good way to hone our tools while enhancing their assurance.

Devices

Device Drivers

Bad device drivers are blamed for many Windows crashes.

The SLAM project used *predicate abstraction* and *model checking* to verify Windows device drivers.

It led to the **Static Driver Verifier (SDV)** toolkit (<http://research.microsoft.com/slam/>).

The main interest in SDV is in verifying the safety of device drivers.

The main challenge is scalability in terms of language features program size, and analysis speed.

Medical Devices

Medical devices are obviously safety critical, and there are well known examples of fatal software bugs (Therac 25).

The FDA estimates that of the 3140 medical devices recalled during 1992-1998, 242 were attributable to software failures. (<http://www.fda.gov/cdrh/comp/guidance/938.html>)

Infusion pumps

(http://en.wikipedia.org/wiki/Infusion_pump) deliver drugs to patients and are often used in battlefield situations.

The CARA infusion pump has been proposed by Dave Hislop as a verification challenge

(<http://bsd7.starkhome.cs.sunysb.edu/~cara/>).

Systems: Lightweight Runtime

This challenge was proposed by Helen Gill.

The idea is to have a verified lightweight RTOS for embedded applications.

The runtime supports a virtual machine for secure, real-time applications with a file system, database, networking support, and high-level programming capability.

Systems

Systems: Hypervisor

A *hypervisor* (<http://en.wikipedia.org/wiki/Hypervisor>) supports multiple operating systems on the same hardware, e.g., VMware, Xen, without interference.

Hypervisors virtualize OS-level tasks.

Paravirtualization requires the operating system to be modified.

Many system architectures such as those for avionics or security applications require hypervisors to ensure noninterference.

Systems: Operating Systems Security

Goal: Verifying the information security of SELinux (<http://www.nsa.gov/selinux/>) or FreeBSD.

The functionality would include the file system, authentication, access control, mail, firewall/VPN, TCP/IP, and buffer overflows.

Applications

Applications: E-Voting

Electronic voting (<http://www.eff.org/Activism/E-voting/>) (<http://www.sos.cs.ru.nl/research/society/voting/index.html>) presents a challenge for system design as well as verification.

E-voting is the focus of NSF's ACCURATE program.

The Australian Capital Territory Electoral Commission makes its e-voting source code available (<http://www.elections.act.gov.au/Elevote.html>)

The Dutch source code is also partly open.

The requirements include: a vote can be cast for any eligible candidate, the outcome of an election must correspond to the votes cast, and votes are verifiable while preserving anonymity.

Conclusions

Large-scale verification experiments are needed to drive the research and to demonstrate a compelling value proposition.

These challenges need to be framed so as to accommodate diverse approaches to the verified code.

Most of the experiments will involve multiple tools. The scalability of these tools will be tested.

Inverting Gresham's law: *Bad software lives forever. Good software gets updated until it goes bad, in which form it lives forever. Casey Schaufler*

Can we make good software proliferate?