SPOPS: Secure Operating Systems for Trusted Personal Devices

Gilles Barthe

INRIA Sophia-Antipolis, France

http://www.lifl.fr/RD2P/SPOPS/

Setting

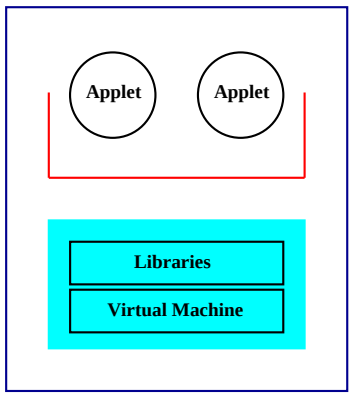
ICT infrastructures are evolving into huge distributed networks of smart devices:

- *flexibility:* aimed at providing seamless access to located services,
- *heterogeneity:* devices may greatly vary in connectivity, computational power, libraries, etc.
- *extensibility:* possible to modify or enhance the computational infrastructure over the network (remote maintenance), or able to upgrade itself by fetching off-the-shelf components (self-healing or self-evolving system)
- *interactivity:* possible to delegate some tasks (computation, storing) to other devices
- *security:* devices and applications are subject to stringent security constraints (confidentiality, integrity, availability, privacy)

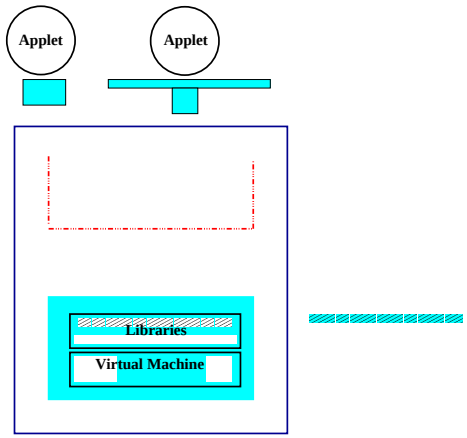
Beyond the sandbox model

We focus on very large networks of JVM-enabled devices without making two common assumptions:

- distinction between TCB and untrusted applications
- idea of central trust authority



Ideal scenario



Scenario considered

Need of expressive formalisms for security policies.

Objectives

- Develop methods to enforce standard security properties: confidentiality, availability
- Propose a security architecture to enforce customizable security policies in such a model
- Experiment with the architecture to validate components of Java-based operating systems

Confidentiality and availability

- Confidentiality:
  - Sound information flow bytecode verifier for sequential JVM
  - Type-preserving compilation for a fragment of JFlow.
  - Program transformation to avoid timing leaks.
  - Logical characterizations of non-interference.
- Availability: we focused on two resources, i.e. CPU and memory.
  - Provided WCET calculus for embedded code, with pre-calculation off-device.
  - Provided logical characterizations of memory usage.

Proof Carrying Code

Programs are equipped with certificates, i.e. mathematical proofs that they obey their specification, which are verified automatically at the consumer side by a proof checker:

- No need to trust the code producer nor the compiler
- Transparent to the code consumer (no run-time penalty, no proof-search)
- Versatile (covers a wide range of safety policies)

Which PCC?

- In the context of mobile and embedded code, correctness guarantees must be given for compiled programs
- There is currently no mechanism for bringing the benefits of source code verification to code consumers
- The objective of our work is to build a mechanism that enables to exploit the results of source code verification for checking compiled programs

Certificate Translation

Definition (Certificate Translation)

Mechanism that allows transferring evidence from source programs to compiled programs (i.e. translating certificate of source programs into certificates of compiled programs)

Remarks:

- Certificate translation is not certified compilation, nor certifying compilation.
- Certificate translation is relevant for interactive and automatic verification.

Towards Certificate Translation for Java

The proof environment JACK has been extended to support:

- Compilation of JML annotations
- Weakest Precondition Calculus at bytecode level

We have shown that non-optimizing compilers enjoy (almost-) preservation of proof obligations, an instance of certificate translation where the translation of proofs is the identity.

Applications

- JITS:
  - JACK has been used to show exception safety for TCP/IP and scheduler.
  - Annotations on bytecode have been used to optimize bytecodes.

  Joint work between A. Courbot and G. Grimaud (LIFL), J.-L. Lanet and M. Pavlova (EVEREST), and J.-J. Vandewalle (Gemplus).
- Memory: Annotations on bytecode have been used to specify and verify resource consumption of Java applets.

Directions for future work

- Combine type-based and logic-based analyzes for information flow and resource usage
- Extend JACK with certificates and implement certificate translation
- Validation of functional properties of components
- Optimizing compilers and OS components written in DSLs