Enforcing Security and Safety Models with an Information Flow Analysis Tool

Rod Chapman
Praxis High Integrity Systems

Contents

The problem…

- Building multi-level security- and safety-critical systems
- We may need **multiple** integrity levels in a single software system.
- We wish to **partition** these to ease verification.
- We want **analysis** to verify partitioning and other properties and/or policies.
  - E.g. "no leaks"

The problem (2)

- Why not just verify everything at the highest integrity level?
  - May be prohibitively expensive!

- Example integrity classifications
  - Common Criteria: EAL 1 - 7
  - UK Def Stan 00-56: SIL 1 - 4
  - DO-178B: levels E - A
  - and many more…

The big idea…

- Why not use an **information flow analysis** (IFA) framework to verify separation of multiple safety and/or security levels?
- Not a new idea!
  - See Denning/Denning paper from CACM 1977…
- SPARK is unique in providing a **decidable** and **sound** IFA facility.

Implementation

- SPARK already supports the notion of an **own variable** declaration - an "announcement" that a package contains persistent state.
- If we knew the "Integrity" of these variables, then we could check the information flow between them.

- An extension of the "own variable" annotation is therefore needed.

Implementation (2)

- Own variable annotation is extended with a new Integrity property. Argument is a Natural number (an ordered, discrete type…)

- Typed constants can be used to declare named Integrity levels.

Implementation (3)

```
package Ejector_Seat
--# own out Fire (Integrity => SIL4);
is
   procedure Panic;
   --# global out Fire;
   --# derives Fire from ;
end Ejector_Seat;
```

Implementation (4)

- A subprogram declaration gives the **derives** relation for that subprogram.
- If this shows info flow from A from B, and both A and B have a well-defined Integrity level, then that flow can be checked right there.
- If A or B is a formal parameter, then check at each call site, **after** substitution of actual for formal parameters.

Implementation (5)

- Example policies
- Security: Bell/LaPadula
  - "Write Up" is OK (e.g. Secret may write Top Secret)
  - "Write Down" NOT OK - no Top Secret data going to an Unclassified output please!
- Safety: non-interference - SIL1 inputs should not affect SIL4 outputs.

Implementation (6)

- What about variables that don't have a specified integrity?
- Assume the worst! For example:
- For safety: Assume all inputs are untrusted. Assume all outputs are safety-critical.
- For security: Assume all inputs are Top Secret. Assume all outputs are Unclassified.

Case Study: SHOLIS

- SHOLIS: Mixed SIL system: some SIL4, some "not SIL4" functions.
- Separation originally argued manually based on SPARK IFA and other analyses. Manual checking of non-interference was time-consuming.
- Can we automate this?

SHOLIS (2)

- Method:
  - Classify and add Integrity property to all own variables.
  - Analyse with new Examiner and check results against those obtained manually.

SHOLIS (3)

- Results:
  - 123 SIL4 Variables, 110 Non-SIL4.
  - In whole program, only ONE case of interference between the two…
  - The Display output buffer - all display data "merges" here prior to transmission to displays.
  - This case was known and expected.
  - All other information flows OK.

Future work

- Expand and complete implementation
  - State refinement
  - Constants with Integrity levels.
- Multi-dimensional Integrity - safety **and** security at the same time?!?
- "Smart Certification"
  - Classify Integrity of **packages** and **subprograms.**
  - Support and verify the partitioning of verification effort.

Questions?

Resources

- Me: rod.chapman@praxis-his.com
- SPARK: www.sparkada.com
- Praxis: www.praxis-his.com