On High-Assurance Information-Flow-Secure Programming Languages (Extended Abstract)

Toby Murray
NICTA and the University of New South Wales, Sydney, Australia
Email: toby.murray@nicta.com.au

Early work on information flow security sought to develop theories for proving the absence of unwanted information leakage in *high-assurance systems*, like those that process classified data. Decades later, modern security-critical systems are more prevalent, face greater security threats, but are rarely formally proved to be information-flow secure, not least because doing so remains fairly expensive [5].

Information-flow-secure programming languages, like Jif, JSFlow, LIO and Paragon, offer hope for reducing the cost of building information flow secure systems. However, they are ill-suited to building formally verified high-assurance systems because each has an overly large trusted computing base (TCB). For instance, Jif and Paragon both rely on Java, so their TCB includes not only their compiler but also the Java TCB — which in 2002 comprised anywhere upwards of 50,000 to 230,000 lines of unverified code [1].

We argue that high-assurance systems demand high-assurance information-flow-secure programming languages. The compiler for such a language shouldn't have to be trusted. Instead, its output should be automatically formally certified as being secure. Recognising that security is the overriding concern for these systems, such a language can also eschew general-purpose language features to reduce its TCB, and ease the certification of its compiler-produced output [2].

Such languages must handle the concurrency and dynamism of modern high-assurance systems, and allow compositional security reasoning with assumptions. Consider a *dual-personality* smartphone whose *classified* personality allows the user to send and receive classified information that is never revealed outside this personality. Figure 1 contains a simplified fragment of a hypothetical *input driver* component, which directs user input to the currently active personality. Input arrives via the `input` variable, and is copied via the `temp` variable to one of two *input buffer* variables, `low` and `high`, depending on which personality is active, stored in the `cur_pers` variable. `input` is updated by some other concurrently running component when new input is available; `cur_pers` is updated when the user switches personalities.

Here, the classification of the data held by the `input` variable varies dynamically. At any point in time, its classification is determined by the `cur_pers` variable: `input` is classified Low iff `cur_pers` is zero, and is High otherwise. Thus `input`'s classification is *value-dependent* [3, 6].

The comments encode assumptions that this code makes to be correct. It assumes that no other component will (1) modify `input`, which implies not changing its classification by modifying `cur_pers`; or (2) modify or read `temp`, which allows `temp` to be safely classified Low always [4].

```
// assume: NoWrite input
// assume: NoReadOrWrite temp
temp = input;
if (cur_pers == 0)
   low = temp;
else
   high = temp;
temp = 0; // clear temp
```

Fig. 1. A snippet of a dynamic input driver component.

As a step towards information-flow-secure languages for high-assurance systems, we extend [4] to yield the first theory of concurrent, value-dependent information flow security that supports compositional reasoning with assumptions. We then provide a roadmap for developing self-certifying high-assurance information-flow-secure programming languages.

Acknowledgements

NICTA is funded by the Australian Government through the Department of Communications and the Australian Research Council through the ICT Centre of Excellence Program.

References

[1] A. W. Appel and D. C. Wang, "JVM TCB: Measurements of the trusted computing base of Java virtual machines," Princeton University, Tech. Rep. TR-647-02, 2002.

[2] G. Keller, T. Murray, S. Amani, L. O'Connor-Davis, Z. Chen, L. Ryzhyk, G. Klein, and G. Heiser, "File systems deserve verification too!" in *PLOS*, 2013, pp. 1–7.

[3] L. Lourenço and L. Caires, "Dependent information flow types," in *POPL*, 2015, pp. 317–328.

[4] H. Mantel, D. Sands, and H. Sudbrock, "Assumptions and guarantees for compositional noninterference," in *CSF*, 2011, pp. 218–232.

[5] T. Murray, D. Matichuk, M. Brassil, P. Gammie, T. Bourke, S. Seefried, C. Lewis, X. Gao, and G. Klein, "seL4: from general purpose to a proof of information flow enforcement," in *S&P*, 2013, pp. 415–429.

[6] L. Zheng and A. C. Myers, "Dynamic security labels and static information flow control," *International Journal of Information Security*, vol. 6, no. 2–3, 2007.