R209 Essay: Correctness vs. Mitigation

Chongyang Shi (*cs940*)

November 20, 2017

1 Summaries of research

The paper by Klein et al. [1] discussed the design and construction process of a formally verified general-purpose L4 microkernel, *seL4*. In order to ensure the functional-correctness of the microkernel, thorough care had been taken by the authors to design seL4's programming model while minimising the introduced complexities in verification. A three-level refinement process was used – a Haskell-based executable model implements the full functionality of the microkernel's abstract specification, which is then optimised for performance in a C implementation. Isabelle/HOL was used in an interactive verification process to ensure functional-correctness. The resulting microkernel is comparable in performance to other L4 kernels, while imposing a reasonable materialistic cost in verification. As a potential improvement to the paper, more discussions on the impact of design decisions on the programmer's ability to write concise and efficient code for the kernel could be included.

The article by Bessey et al. [2] introduced notable observations and unique challenges faced by the authors in the process of commercialising a static analysis-based bug-finding tool. Their product applies an unsound traversal on the target codebase to find as many errors as possible. After the product was adapted for use on large commercial codebases, caveats including as integrating with different build systems, dealing with compilers and runtime environments not compliant with language standards, and working with misinformed users as well as commercial demands were identified by the authors, along with some attempts to address these challenges. A possible extension to this paper is discussion on whether code written in languages with tighter ecosystems such as Apple's Objective-C are easier for static analysis in the field.

The survey paper by Szekeres et al. [3] summarised common mitigations against memory corruption bugs, as well as performance overheads, compatibility issues, and attack vectors associated with each mitigation technique. Starting by introducing common types of memory corruption attacks, the authors first gave an overview on protection systems in use based on both probabilistic protection and deterministic protection. After discussing important metrics as well as tradeoffs in cost and compatibility, they further described how current systems measure in these aspects. They also covered defences against generic attacks and control-flow hijack attacks. The authors concluded by highlighting performance and compatibility constraints in current systems which prevent their wide adoption. A critique for this paper concerns on its lack of clarity in structure, such as the repeated description and discussion of stack smashing protection [3, III, VIII-B].

2 Key themes of research

2.1 Laboratory theory versus real-world practice

2.2 Security versus cost

2.3 Benefits of using an intermediate representation

3 Ideas in current context

4 Literature review

References

[1] G. Klein *et al.*, "sel4: Formal verification of an os kernel," in *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles.* ACM, 2009, pp. 207–220.

[2] A. Bessey *et al.*, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[3] L. Szekeres *et al.*, "Sok: Eternal war in memory," in *Security and Privacy (SP), 2013 IEEE Symposium on.* IEEE, 2013, pp. 48–62.