Comments on NIST SP 800-171B

Bryan Parno
Associate Professor
Departments of Computer Science & Electrical and Computer Engineering
Carnegie Mellon University

In the current draft, Section 3.14 includes three valuable technologies: roots of trust, code signing, and formal verification. It is an excellent idea to include all three of these technologies in the standard, but it may be worthwhile to separate them out, since they all serve complementary purposes, rather than providing alternatives for a single purpose. In particular, roots of trust help ensure your device is running the code you expect and protect secrets that belong to that code. Code signing is one way to leverage a root of trust to establish a chain of trust, but it can (and should) also be used to validate software updates. Both technologies are useful for establishing what software is running (or ensuring that only specific software can run), but neither establishes the software itself is *trustworthy*.

Formal software verification is one of the strongest techniques we know of for establishing the trustworthiness of software, and hence clearly deserves a place in a standards document, such as this, for critical systems. While verification is still a non-trivial undertaking, it is becoming more feasible, with significant advances in the last decade. Verified software artifacts include operating system kernels [3, 5, 10], secure application stacks [4], distributed systems [4, 9], compilers [6, 7], and cryptographic libraries [8, 12].

Furthermore, several studies have empirically confirmed that verifying software results in qualitatively better code compared with traditional software development practices [1, 2, 11]. These studies found numerous defects and vulnerabilities in traditional software, but found far fewer in verified systems. Indeed, they failed to find any in the verified software itself. The only bugs found in verified systems were in the much smaller and simpler specifications, which can can more readily be audited and tested than the software itself.

Nonetheless, since verification is still costly, the current language (preferring verified software when available) sets a reasonable standard. Such standards will, in turn, help raise the bar for software running on our critical infrastructure. The other two technologies (roots of trust and code signing) are already standard and ubiquitous in modern commodity systems, and hence should be mandatory in critical systems considered by SP 800-171. Hardware-based roots of trust, such as the Trusted Platform Module, are found in hundreds of millions of laptops and desktops, and are standard issue on most smart phones and tablets, where they are used to implement secure boot procedures. Similarly, code signing is standard for all major operating systems, BIOS updates, and mobile app updates.

At a lower level, the first paragraph of Section 3.14 ends with, “protecting the confidentiality of the key used to generate the hash; and using the public key to verify the hash information”. In both places where it says “hash”, it should say “signature”. The standard definition of a hash algorithm takes a single input (the data to be condensed); it does not take a key. A signature algorithm will often internally use a hash function, either for security, efficiency (so that expensive asymmetric operations need only operate over a small, fixed amount of data) or both.

References

[1] K. Fisher, J. Launchbury, and R. R. . doi:10.1098/rsta.2015.0401. The HACMS program: Using formal methods to eliminate exploitable bugs. *Philosophical Transactions A, Math Phys Eng Sci.*, 375(2104), Sept. 2017.

[2] P. Fonseca, X. W. Kaiyuan Zhang, and A. Krishnamurthy. An empirical study on the correctness of formally verified distributed systems. In *Proceedings of the ACM EuroSys Conference*, Apr. 2017.

[3] R. Gu, J. Koenig, T. Ramananandro, Z. Shao, X. N. Wu, S.-C. Weng, H. Zhang, and Y. Guo. Deep specifications and certified abstraction layers. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, 2015.

[4] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, and B. Zill. Ironclad Apps: End-to-end security via automated full-system verification. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, October 2014.

[5] G. Klein, J. Andronick, K. Elphinstone, T. Murray, T. Sewell, R. Kolanski, and G. Heiser. Comprehensive formal verification of an OS microkernel. *ACM Transactions on Computer Systems*, 32(1), 2014.

[6] R. Kumar, M. O. Myreen, M. Norrish, and S. Owens. CakeML: a verified implementation of ML. In *Proceedings of the ACM Symposium on Principles of Programming Languages (POPL)*, Jan. 2014.

[7] X. Leroy, S. Blazy, D. Kästner, B. Schommer, M. Pister, and C. Ferdinand. Compcert – a formally verified optimizing compiler. In *Embedded Real Time Software and Systems (ERTS)*. SEE, 2016.

[8] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet, T. Ramananandro, A. Rastogi, N. Swamy, C. Wintersteiger, and S. Zanella-Beguelin. EverCrypt: A fast, verified, cross-platform cryptographic provider. Technical Report 2019/757, ePrint, July 2019.

[9] J. R. Wilcox, D. Woos, P. Panchekha, Z. Tatlock, X. Wang, M. D. Ernst, and T. Anderson. Verdi: A framework for implementing and formally verifying distributed systems. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, June 2015.

[10] J. Yang and C. Hawblitzel. Safe to the last instruction: Automated verification of a type-safe operating system. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, 2010.

[11] X. Yang, Y. Chen, E. Eide, and J. Regehr. Finding and understanding bugs in C compilers. In *Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI)*, June 2011.

[12] J. K. Zinzindohoué, K. Bhargavan, J. Protzenko, and B. Beurdouche. HACL*: A verified modern cryptographic library. In *ACM Conference on Computer and Communications Security*, 2017.