Epilogue

The End. Or perhaps you prefer: *And They Lived Happily Ever After.*

But our story is not so simple. We are closer to the beginning than the end.

In this closing commentary—in contrast to the rest of the book, which aimed to present generally accepted facts and consensus views—we include also personal views and opinions, warning that these may change as we learn more and environments evolve.

Having read major portions of this book, you now have a solid background: you have learned some key approaches and principles to help build security into systems, you have a better understanding of what can go wrong, and you are better able to recognize and mitigate risks in your own use of computer systems. As new security students are told: we must learn to walk before we can run. If you have read this book—ideally, as part of a course supplemented by hands-on, programming-based assignments—you are now at walking speed. Do you know everything there is to know about computer security and the Internet? It is my duty to now inform you that this is not the case.

We have covered quite a bit of ground. But most of it has involved relatively small, individual pieces—important basic mechanisms for security, applications highlighting how such tools have been applied, and pointers into the literature (a few of which you followed, if you were keen). Which of these are standard *tools*, and which are the *jewels*, depends in part on personal perspective. Chapter 1 ended by considering: “Why computer security is hard”. We now have better context from which to pursue this question, but rather than return to elaborate one by one on the items noted, we selectively consider a few issues more deeply, and as usual, provide a few stepping-stone references into the literature.

HUMAN FACTORS. Security experts in academia typically have a primary background in mathematics, computer science or engineering. Only in the past 15 years has it become more widely appreciated that expertise from the fields of psychology and cognitive science is of critical importance to understand how usability affects security, and vice versa. How people think and make security-related decisions when using computer systems—involving *human factors* issues—is more difficult to predict than purely technical elements. Traditional formal analysis methods are typically unsuitable here—there is a disconnect between how we behave as humans, and the tools historically used to reason about technical systems. Some experts believe that the stronger technical protections become, the more we will see *social engineering* as a non-technical attack vector. This book has only scratched the surface of usable security, e.g., in discussing passwords, *phishing* and web *security indicators*. Beyond the references suggested in the Chapter 9 end notes, Norman [10] is recommended as an accessible source on usability. Many software developers will benefit from learning about *heuristic evaluation* [9] and *cognitive walk-through* [14], two lightweight usability evaluation methods, often used as precursors to more time-consuming formal user studies.

MODELS VS. REALITY. Models, briefly discussed in Chapter 1, are tremendously useful for design and analysis. It turns out that people, including security researchers, often mistakenly believe that properties proven about abstract models will necessarily hold true for the real systems modeled. This is false due to the limitations of models, as clearly explained by Denning [4], and more recently Herley [6]. A key observation is that attacks in practice are often outside of a model's assumptions. Therefore, "proofs" of security are misleading—it is not that the logical arguments are incorrect, but that they focus narrowly on specific properties, and depend on assumptions that fail to hold in actual systems. Some experts argue, in response, that "everybody knows that proofs depend on assumptions and the model", but too often (in our observation), stated results are widely misinterpreted ("the system is secure; hurrah!"), with no one responsible for verifying that real systems match the assumptions or model.

TESTING FOR SECURITY. A major challenge in practice is that we don't have reliable methods for "security testing". As noted in Section 1.6, (complete) testing for the *absence* of exploitable flaws cannot be done by traditional input-output testing—at best, that establishes compliance with known test cases. The (complete) task appears impossible: predict all possible things that an attacker *might* do. This returns us to models: if we explicitly rule something out of a model, that in the real world an attacker might actually do, then the model is incomplete, and likewise if we implicitly forget to include something in the model. Another explanation is as follows (see Torabi Dashti [11] for details). Define Type-I tests to be those that attempt to show that a system fails to meet its *specification* (a description of desired system behaviors); if no such test shows a failure, confidence is gained. Define Type-II tests as those that attempt to show that assumptions about an *adversarial environment* are false (i.e., assumptions about how a target system interacts with an environment that includes an adversary). Now, *functional testing* involves Type-I tests, while *security testing* (testing to meet *security requirements*) involves both types—and is thus strictly harder. Note that the resources and abilities held by an adversary directly impact whether security requirements can be violated by the attacker. In testing, an adversary's abilities are based on assumptions—and thus, so is the answer to whether or not a system meets its security requirements. The next question is (looping back): How do we test whether the assumptions are valid? This remains unanswered, with an asserted conclusion [11] that security testing escapes automation and systematization.

COMPOSITION AND EMERGENT PROPERTIES. Suppose we have a collection of subsystems (components), and by good fortune, have high confidence in the security properties of individual pieces. What can be said about their combination? This raises the issue of *secure composition*. For a given property P, if we combine two components that both have P, a combined system may or may not—and, combining two components that individually do *not* have P might yield a system that does. Under what circumstances are security properties composable? This turns out to be a complex and little understood problem—for an introduction, see Datta [3]. A simpler problem is *secure protocol composition* [2]. Related to this is the concept of an *emergent property* within a system—which by one definition [15], is a property not satisfied by all individual components, but by their composition. Such a property may be problematic (if it enables attacks) or beneficial (if it stops attacks). The state of the art is that we know little about emergent properties in real systems—thus establishing trustworthiness in practice remains largely out of reach. Nonetheless, a starting point is to build real-world components in some manner by which we gain high confidence in selected security properties, e.g., building components that rule out entire classes of known attacks. It is for this reason that real-world systems such as Multics (see Chapter 5 references) and CHERI [13] (mentioned also in the Foreword) are worth examining as detailed case studies.

TRUSTING HARDWARE. As mentioned in the Chapter 5 end notes, the 1972 Anderson report [1] already raised as an issue the need to trust the entire computer manufacturing supply chain. An assumption that is almost always implicit, and rarely acknowledged, is that we assume trustworthy hardware. Distinct from its robustness and dependability, hardware itself may have embedded malicious functionality. A separate hardware issue involves classes of attacks that exploit hardware artifacts resulting from performance optimizations on commodity processors, e.g., leaking sensitive kernel information in cache memory through use of *speculative execution*. These attacks include *Meltdown* (see the end of Section 7.4), *Spectre* [7], and (impacting SGX hardware) *Foreshadow* [12]. These are *side-channel* attacks in that the attack vectors involve non-standard access channels. We now understand that most of today's software runs on commodity hardware that behaves differently than the relatively simple security models assumed until very recently. Attacks are enabled by this gap between a typical programmer's model of their target CPU, and the finer-grained state transitions of actual hardware, which may be viewed as a *weird machine* subject to serious exploitation—as Dullien [5] explains.

ADIEU. This ends our selective tour of issues that complicate security in practice. The details of these, and many other important topics, are not explored herein. It should be clear that our journey is just beginning. I wish you well on your path to enlightenment.

References

[1] J. P. Anderson. Computer Security Technology Planning Study (Vol. I and II, "Anderson report"), Oct 1972. James P. Anderson and Co., Box 42, Fort Washington, PA, 19034 USA.

[2] A. Datta, A. Derek, J. C. Mitchell, and D. Pavlovic. Secure protocol composition. In *ACM Workshop on Formal Methods in Security Engineering (FMSE)*, pages 11–23, 2003.

[3] A. Datta, J. Franklin, D. Garg, L. Jia, and D. K. Kaynar. On adversary models and compositional security. *IEEE Security & Privacy*, 9(3):26–32, 2011.

[4] D. Denning. The limits of formal security models. *National Computer Systems Security Award Acceptance Speech, Oct 1999*. `https://faculty.nps.edu/dedennin/publications/National%20Computer%20Systems%20Security%20Award%20Speech.htm`.

[5] T. Dullien. Weird machines, exploitability, and provable unexploitability. In *IEEE Trans. Emerging Topics in Computing*. Early access 19 Dec 2017 (print version to appear). For history on weird machines, see also: `https://www.cs.dartmouth.edu/~sergey/wm/`.

[6] C. Herley and P. C. van Oorschot. Science of security: Combining theory and measurement to reflect the observable. *IEEE Security & Privacy*, 16(1):12–22, 2018.

[7] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre attacks: Exploiting speculative execution. In *IEEE Symp. Security and Privacy*, 2019.

[8] J. Nielsen and R. L. Mack, editors. *Usability Inspection Methods*. Wiley & Sons, 1994.

[9] J. Nielson. Heuristic evaluation. 1994. Pages 25-64 in [8].

[10] D. Norman. *The Design of Everyday Things*. Basic Books, 1988.

[11] M. Torabi Dashti and D. A. Basin. Security testing beyond functional tests. In *Engineering Secure Software and Systems (ESSoS)*, pages 1–19, 2016.

[12] Van Bulck, J., M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security*, pages 991–1008, 2018.

[13] R. N. M. Watson, R. M. Norton, J. Woodruff, S. W. Moore, P. G. Neumann, J. Anderson, D. Chisnall, B. Davis, B. Laurie, M. Roe, N. H. Dave, K. Gudka, A. Joannou, A. T. Markettos, E. Maste, S. J. Murdoch, C. Rothwell, S. D. Son, and M. Vadera. Fast protection-domain crossing in the CHERI capability-system architecture. *IEEE Micro*, 36(5):38–49, 2016. See also: ASPLOS 2019.

[14] C. Wharton, J. Rieman, C. Lewis, and P. Polson. The cognitive walkthrough method: A practitioner's guide. 1994. Pages 84-89 in [8].

[15] A. Zakinthinos and E. S. Lee. Composing secure systems that have emergent properties. In *IEEE Computer Security Foundations Workshop (CSFW)*, pages 117–122, 1998.