Cybersecurity Architecture

The Design and Security of Modern Computing Systems

Bruce Jacob

with contributions from **William Casey** and **Anthony Melaragno**

United States Naval Academy
Cyber Science Department

Dedication

To my wife Valerie — thank you for giving me the best life imaginable

Contents

Preface

Historically, software engineers and hardware engineers are disparate groups that don't talk to each other. The result is that *computer design* is taught by one group (hardware people), and *computer security* is taught by another (software people). Because these two groups don't interact much, neither knows much of the other's discipline, and therefore *those they teach* learn next to nothing of the other's discipline. This is problematic because computer design and computer security are intimately dependent on each other, and when system designers fail to understand both topics, catastrophe results.

As we will show, two of the more critical security holes known today (buffer overruns and *Spectre/Meltdown*) exist only because system designers unwittingly undermined the very protections that would otherwise have guaranteed absolute insulation from abuse. In addition, because the topics of security and computer design are taught separately, students cannot see the root causes *behind* security failures … this explains why many of the widely proposed and adopted solutions to buffer overruns and Spectre/Meltdown fail to address the root issues.

The division between hardware and software is seen in both education and the workplace. Software and hardware engineers are typically housed in different departments within different organizations, often located in completely different buildings. They don't typically see eye-to-eye. Software people consider hardware people to be unibrow neanderthals who think that clubs, sharpened rocks, and C programming are paragons of high-tech (think the opening scene in Stanley Kubrick's *2001: A Space Odyssey*). Hardware people consider software people to be flighty, self-satisfied pansies who complain far out of proportion to what useful work they accomplish (think Harvey Ko-

rman and Andréas Voutsinas as the bickering French noblemen in Mel Brooks's *History of the World, Part I*).

The division in perspective leads to division in real system designs. Typically, hardware is designed with little regard to the software that will run on it, and software is almost always designed with little regard to the hardware it runs on. For instance, *portability* is considered a laudable achievement in the land of software — and it literally means the writing of software in such a way that it is ***not*** dependent ***at all*** upon the underlying hardware. Again, we will show how today's biggest security holes come from misalignment between one's hardware and software designs.

We hope that this textbook will help to address the problems that the division has created. We have found that it is a natural fit to teach computer organization/architecture alongside computer security; doing so leads to a natural understanding of the purpose behind hardware and operating-system primitives. We hope that treating the two domains together in the same semester-long course will put an end to the practice of designers unknowingly disabling critical security features for the sake of convenience.

This textbook presents the design and security of microprocessors and the operating systems that run on them. An example computer system is built from the ground up, so that all of the implicit assumptions are visible and identifiable. This course covers the rudiments of logic design, computer organization and architecture, compiler design, and operating systems design, so that the student can participate in the design and development of all of these components. Then it discusses security aspects of the design in depth, so that the student can see the ways in which current designs both succeed and fail to protect against intrusion.

The design projects are presented as incomplete solutions: they are our system-implementation code, with the 'intelligent' portions removed. The security projects are essentially guided tours through the various system behaviors, with the student implementing each of the attacks and learning how one can defend against them.

By the end, the student should have a solid understanding of how current systems are built, why they were designed in the ways that they were, what their weaknesses are, what current mitigation strategies are considered to be appropriate … as well as some thoughts on how one can do better in future designs, both hardware and software. The result should be a relatively comprehensive view on current design and security principles of modern computing systems.

Good luck, and happy coding.
—The Authors, Annapolis