SECRETS OF A CYBER SECURITY ARCHITECT

SECRETS OF A CYBER SECURITY ARCHITECT

Brook S. E. Schoenfield

CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
AN AUERBACH BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

© 2020 by Brook S. E. Schoenfield
CRC Press is an imprint of Taylor & Francis Group, an Informa business

No claim to original U.S. Government works

Printed on acid-free paper

International Standard Book Number-13: 978-1-4987-4199-6 (Hardback)

Author Note

All references to *Securing Systems* throughout the book are from:

Schoenfield, B. (2015). *Securing Systems: Applied Security Architecture and Threat Models.* Boca Raton, FL: CRC Press.

All references to *Core System Security* throughout the book are from:

Schoenfield, B. (2014). Applying the SDL Framework to the Real World. In Ransome, J. and Misra, A. *Core Software Security: Security at the Source,* Ch. 9, pp. 255–324. Boca Raton, FL: CRC Press.

Trademarks Covered in This Book

Dedication

This book is dedicated to the many security architects with whom I've worked and from whom I've learned: mentors, mentees, peers, students, comrades, friends. The InfoSec architecture team at Cisco (circa 2000–2011) and McAfee's Product Security Champions (2012–2018) deserve my special gratitude. Collectively, we've contributed to a discipline known now as "security architecture." Herein lie the many fruits harvested from our discussions, speculation, and philosophizing; the trials and successes we've shared; and your many insights. Thank you.

Contents

Author Note/Trademarks Covered in This Book *v*

Dedication *vii*

Contents *ix*

List of Figures and Tables *xiii*

Foreword *xv*

Preface *xvii*

Acknowledgments *xxiii*

About the Author *xxvii*

Chapter 1 The Context of Security Architecture **1**

1.1 Omnipresent Cyber War 1
1.2 Know the Threat Actors 5
1.2.1 Useful Exploits Don't Die 15
1.3 Everything Can Become a Target 16
1.4 Warlords and Pirates 20
1.5 What Is the Scope of a Security Architect? 22
1.5.1 Are There Really Two Distinct Roles? 22
1.6 Essential Technique 26
1.6.1 Threat Modeling: An Essential Craft 26
1.6.2 Architecture Is Primary 28
1.7 Aiming Design Toward Security 29
1.7.1 What Is Secure Software? 29
1.7.2 Secure Design Primer 30
1.8 Summary 31

Chapter 2 What Is Security Architecture, and Why Should I Care? 33
2.1 Define Security Architecture 33
2.1.1 Software Security 34
2.1.2 Security Architecture Practices 34
2.2 Relevant Knowledge Domains 35
2.3 More About Architecture 38
2.4 Architectures of Security 40
2.5 Architecture as a Part of Cyber Security 41
2.6 Security Architecture in Software Development 42
2.7 Generally, Experience Is a Teacher 44
2.8 Introducing Attack Methods 45
2.9 Speaking of Defense 47
2.10 More Precise Definition 48
2.11 Summary 49

Chapter 3 Architecture, Attacks, and Defenses 51
3.1 Yes, Exploit Details, But 51
3.2 Security Architects Must . . . 52
3.3 Understanding Categories of Attacks 53
3.4 Attack Knowledge for Defense 55
3.5 Example: Heartbleed Analysis 57
3.5.1 Heartbleed Technical Analysis 60
3.6 Analyze to Defend 66
3.7 Turn Off TLS? 69
3.8 Security Architecture Analyses 70
3.8.1 Some Cheap Risk Concepts 72
3.8.2 JGERR Risk Rating 72
3.8.3 At Base: Threat Model 73
3.9 Threat Modeling Definition 73
3.9.1 Alternate Definition 74
3.9.2 When Is My Threat Model Done? 74
3.10 Summary 76

Chapter 4 Culture Hacking 79
4.1 Team Tourism 79
4.1.1 Build and Maintain Trust 81
4.1.2 Don’t Squander Influence 84
4.2 Threat Modeling: Just Do It 86
4.2.1 “Trust Developers?” 87
4.2.2 Threat Model Training Is for Everyone 88
4.3 More Culture Hacks 89
4.3.1 Nimble Governance 90
4.3.2 Build Skills by Sharing 93
4.3.3 What to Do About “It Depends” 93

4.3.4 Is the Threat Model Finished? 94
4.3.5 Create a Security Contract 97
4.3.6 Threat Models Are Not Additive! 99
4.3.7 Audit and Security Are Not the Same Thing 100
4.4 From Program to Transformation 103
4.4.1 Pro-Social Modeling 103
4.4.2 Leaders Must Get Challenged 104
4.4.3 Hack All Levels 105
4.4.4 Coding Is Fraught with Error 108
4.4.5 Effective Secure Coding Training 109
4.4.6 Make Validation Easy 110
4.5 Summary 112
4.5.1 We All Can Use Some Feedback 114

Chapter 5 Learning the Trade 115
5.1 Attack Knowledge 116
5.2 Which Defenses for What System? 123
5.3 Threat Modeling: The Learning Method 126
5.3.1 How to Escalate for Management Decision 128
5.4 To Accelerate: Cross Pollinate 130
5.5 Build a Community of Practice 131
5.6 Support Learners' Errors 134
5.7 Facilitate as Much as Lead 137
5.8 Summary 137

Chapter 6 Problem Areas You Will Encounter 141
6.1 What Does a Mature Practice Look Like? 141
6.1.1 Do We Add Value? 141
6.1.2 The War Is Over 142
6.1.3 Optimum Tool Use 142
6.1.4 You Know That You're Maturing When 144
6.1.5 "Nothing Proves Architecture Like Nothing" 145
6.1.6 Get It in Writing! 147
6.2 Typical Problems Programs Encounter 147
6.2.1 Scale 147
6.2.2 Assessments Take Too Long 150
6.2.3 Late Engagement 156
6.2.4 Skill Churn 161
6.2.5 Exceptions 163
6.2.6 Fostering Innovation 164
6.3 Dealing with Chaotic Elements 166
6.3.1 There Are Differences 167
6.3.2 Translate and Generalize 168
6.4 Summary 170

Appendix A **Heartbleed Exposure, What Is It Really?** **173**

Appendix B **Developer-Centric Security** **177**

Appendix C **Don't Substitute CVSS for Risk: Scoring System Inflates Importance of CVE-2017-3735** **179**

Appendix D **Security Architecture Smart Guide** **185**

Appendix E **Threat Modeling's Definition of Done** **203**

References **207**

Index **213**

List of Figures and Tables

Figure 5.1	MITRE ATT&CK™ Matrix	119
Figure 5.2	CAPEC Mechanisms of Attack	121
Figure 5.3	ATT&CK Headers Combined with CAPEC Categories	121
[Figure 11.3	Multitenant Data Encapsulations (from *Securing Systems*)]	169
Table 1.1	Summarized Threat Attributes	13
Table 1.2	High-Level Threat Agent Attribute Matrix	14
Table 4.1	Summation of Actions Described in This Chapter	112
Table 5.1	Organization Hierarchy and Potential Impact Relationship	130
Table 5.2	Summation of Actions Described in This Chapter	138
Table 6.1	Summation of Actions Described in This Chapter	170

Foreword

This week's news includes that indicted Iranian hackers are still hard at work, a disruptive cyber event impacted the US power grid, and a high-school dropout hacked a million devices. It included brand-name companies in the headlines for security failures. And it looked very much like other weeks.

Many of the systems whose security failed had compliance checklists. Products were subjected to penetration testing, or ethical hacking, or dynamic testing. And they still failed, because the security approach their creators used didn't address the unique requirements of the systems being built and deployed.

Security architecture is a set of structures for thinking about what we're working on, what can go wrong, and, most importantly, what are we doing about them? We're infusing security into the things we work on. We're building features and worrying about the properties. The distinction is like this: a deadbolt is a security feature, and the steel it's made from has properties. If the steel of the deadbolt is brittle; if the receiver isn't well mounted; if the doorframe is weak, then the deadbolt will not deliver on the security goals for the door. (Incidentally, my book on threat modeling is largely focused on what can go wrong, and as such complements this one.)

Our software, like other things we build, has many interfaces to the world, and attackers can pick and choose which ones to examine or attack.

For years, security has shown up as designs are finalized, and then complained about those designs. It hasn't delivered the security that our society needs, and it hasn't led to effective collaboration.

Architecture, like security, has a bit of a bad name amongst software engineers. Too often, it's people who can't code, won't make tradeoffs, and don't ship, but do object—endlessly. Like any aspect of software, architecture and security can be done well or poorly.

The book that you hold in your hands is about doing it well, but that's not quite right. I don't care much about doing it well, in and of itself, and I don't think Brook does either. I care about helping the people I'm working with make better products, which includes shipping and includes shipping with appropriate security. Let me be clear about what *appropriate* means here: it means that the folks who make product decisions have the information they need to make those decisions. Sometimes you're happy with the result, other times not, but you shouldn't be surprised by the security problems systems have once they ship.

Of course, I prefer to make things I can be proud of, and so I want to do things well, and I hope you want the same. This book is about how to do that for security.

Adam Shostack
September 17, 2019
Seattle, WA

Preface

Context

As I wrote in *Securing Systems, "It is a plain fact that as of this writing, we are engaged in a cyber arms race of extraordinary size, composition, complexity, and velocity." Securing Systems,* p. 5

There are more than three billion[*] people who use the Internet and whose lives are intertwined with their digital devices. These connected people's medical, financial, and other personal data is spread out over hundreds if not thousands of systems run by a multitude of organizations, many of whom do not necessarily have the data owners' best interest at heart. Pandora's[†] cyber box has long since been opened, and the box's "demons" have been loosed upon the digital world. Indeed, most of us participants are attempting to get on with our lives, making our way through the digital battle zone. A few of you readers may actually be engaged in the cyber war in some professional or other capacity. But for the vast majority of us cyber-citizens, we are the collateral damage to conflict that has little to do with us on a personal level.

Like most war zones, in addition to the combatants, there always seem to be those out to make a profit amid the chaos. There be pirates and warlords on these Internet seas, Matey. I think that the current state of affairs may be comparable to the 300 or so years when international commerce was highly dependent upon sea trade, approximately from the late 1500s to the middle of the 19th century, the so-called "golden age of piracy."

Not that we don't have sea pirates today; of course we do. But, during the golden age of piracy, shipping, which, as the main form of transport underpinning international commerce, was never safe from marauding pirates. City states were funded through piracy; piracy made significant contributions to the tax basis of several nations. Piracy, or more properly, the

[*] According to the United Nations population estimates, in 2020, there will be near to 7.8 billion people on planet Earth. Please see https://population.un.org. Estimates of Internet users vary from 3.2 to 4.4. Please see https://en.wikipedia.org/wiki/Global_Internet_usage and https://internetworldstats.com/stats.htm for examples of estimates.

[†] Pandora, the Greek mythological character, not Pandora™, the music streaming company.

privateer, served as a way to fortify a nation's navy in times of war. At the same time, privateers provided a needed boost in state revenue at the expense of a nation's enemies.[*]

The age of piracy seems entirely analogous to the Internet Age's quasi-state "cyber armies" who attack, and sometimes plunder, digital commercial interests. As Dmitri Alperovitch so wryly noted in 2011,

"I divide the entire set of Fortune Global 2,000 firms into two categories: those that know they've been compromised and those that don't yet know."[†]

In other words, any organization with valuable data has been and will be attacked, probably successfully, at some point and with some damage. And, don't all digitally connected organizations have at least some data that can be considered "valuable"? If the data is not intellectual property on which revenue is based, then the personal records of customers or employees will be valuable to digital attackers. For so-called non-governmental organizations (NGOs), strategic plans or the names of operatives in countries in which any disagreement with the local government raises suspicion or even sanction may be considered valuable to those charged with preventing dissent. Can we declare that in the Age of the Internet, no data is safe, no data is without value to attackers? On some days, this may seem to be close to a truism. Isn't this comparable to the golden age of piracy, when no ship in transit was safe?

There's a great deal of money to be made through theft and fraud of one kind or another on the Internet, which I liken to the great oceans of times past. And, there are large organizations and single actors who understand that "there's a sucker born every minute"[‡]—that is, a few billion unwitting potential victims to fleece. A great advantage for these attackers is that, mostly, the pirate does not have to come into physical contact with the victim. Many, if not most, successful attacks are relatively[§] anonymous. Which, one has to admit, is a big advantage for attackers.

We, the Internet connected,[¶] are the targets of all this cyber attack activity rather constantly and continuously.

Of course, government cyber armies practice defensive maneuvers. That is to be expected. Unfortunately, most government defense efforts are not focused on protecting you and me or the many independent or commercial organizations that hold our data as we blissfully go about our digital lives. In the USA, where I live, certainly the federal government is highly concerned about protecting not only its own resources, but also the nation's critical infrastructure. Still, despite the deep concern, most of that actual day-to-day protecting is done by the organizations, public and private, who run the infrastructure, not by the government. You may have

[*] This historical reference is drawn from my reading of historical analysis presented within *Pirate Utopias: Moorish Corsairs & European Renegadoes* (Wilson, 2004).

[†] Alperovitch, 2011.

[‡] Inconclusively attributed to P. T. Barnum.

[§] As of this writing, retrieving any identifying information from sophisticated attacks requires expert forensic analysis. The information retrieved is often quite piecemeal, at best. There may be pointers to the identities of attackers, but often there is no direct link from attack to person.

[¶] About half or so of Earth's human inhabitants do not use the Internet at the time of this writing. I always try to bear this in mind.

read newspaper accounts about the difficulties and gaps in those protections. Sadly, my professional opinion is that these media descriptions are more or (often) less accurate. I'm willing to bet that the USA's enemies are keenly aware of the situation. Many nations' infrastructure is equally at risk; the USA is not alone in this.

Cyber Defenders

To whom or what may a non-technical cyber-citizen turn?

During my 30+ year high-tech career, so called "computer security," "information security," or "cyber security" has grown from a fledgling group of interested amateurs who tried to squeeze in some security work alongside their busy, high-tech day jobs into a full-blown industry of thousands of professionals.

Many security companies have tried to steer clear of jingoistic association. The big vendors often have customers in any number of governments, some of whom at any particular moment will be in active cyber conflict with each other. It seems an interesting turn of events that security companies have to maintain a certain level of cyber neutrality in order to succeed. It turns out that refusing to take sides is just good security business.

Computer security companies offer a dizzying array of technologies, products, and services, many of which are marketed through some variation of "solves your security problems." Unfortunately, not.[*]

Part of the problem is that "security" is ill-defined and highly overloaded. Most security products handle some portion of the computer security challenge. No product I know of comes close to being a total package solution, taking care of all an organization's or person's computer security needs, soup to nuts. Why? Because security is a big, messy, multivariate, multidimensional arena. A reasonable "defense-in-depth" requires many technologies; smart, highly skilled people; and deep and broad analysis—all of which must come together into some sort of functioning whole, that whole often termed "a security architecture."

Why "Security Architecture"

Architecture:

1. The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution.[†]
2. The structure of components, their inter-relationships, and the principles and guidelines governing their design and evolution over time.[‡]

[*] Full disclosure: As of this writing, I work for one of these "security companies." My comments do not represent the opinions of my employer.

[†] *Source:* ISO/IEC/IEEE 42010:2011

[‡] The Open Group, n.d.

"Traditionally, security architecture consists of some preventive, detective and corrective controls that are implemented to protect the enterprise infrastructure and applications."[*]

"It is not possible to build a space ship without vision, goals, and specific objectives, which are expressed in large-scale and also highly specific plans. Cyber security is much the same thing, which is where the practice of security architecture comes into play.

"One definition of security architecture might be, 'applied information security.' Or perhaps, more to the point of this work, security architecture applies the principles of security to system architectures." Securing Systems, p. 14

My previous definition probably assumes far too much to be useful unless one is already familiar with information security and the practice of architecture. By "system architectures" I meant to include any system that requires digital security, be it the architecture of a discrete application or program, a library or application programming interface (API), a global cloud, an organization's digital assets, etc. Though my definition may have sufficed for the purposes of a book on threat modeling, it still feels a bit too opaque, and I fear that it excludes some important aspects.

This book is about security architecture. My goal is to refrain from theory and focus instead on practice. You will not find much theory in this book. I hope that I have provided just enough theory to place the materials into sufficient context for full understanding. For a deeper explanation, I point the reader to any number of books on security architecture, including my own, *Securing Systems,* although that book was also intended to remain grounded in the practical and proven rather than being overly theoretical.

Good security architects have dozens of tricks of their trade in their kips. Herein, you will find my tips and tricks, as well as myriad tried and true bits of wisdom that my colleagues have been gracious enough to share with me.

I want to give these to you, the practitioner, to ease your way. This work can be hard, complex, certainly frustrating. Seasoned architects know how to surmount individual, team, and organizational resistance. They know how to express security requirements in ways that will make the requirements more palatable and, thus, get them accomplished.

Great security architects tend also to be masters of compromise, negotiation, and conflict management. In general, the people for whom I have the highest respect are consummate collaborators, as interested in understanding as providing their own solutions. The vast majority of practitioners with whom I've had the privilege and honor of working have been and continue to be high-integrity individuals who want the best for their organizations and typically, also, through their work, for the world at large. Of course, each of us gets to define "best" for ourselves.

Book Contents

The first chapter of the book is focused on what security architecture is and the areas of expertise a security architect will need in practice. The second chapter delves into the relationship

[*] Ghaznavi-Zadeh, 2017.

between attack methods and the art of building cyber defenses. Security people must become familiar with the many different ways that human actors attack systems. For some branches of digital security, the understanding of attack mechanisms must be deep and thorough. For other areas—say, people managers and executives—perhaps only a glancing understanding of attack methods is required, although often, these roles require a thorough understanding of the potential organizational harm that may result from successful attacks of one kind or another.

For the security architect, the level of understanding will be more holistic. Each class of attack involves a particular type of technical manipulation. Security architects typically must have enough computer science background to understand the basic mechanism of manipulation. Alongside this, an effective security architect will further have a command of the mechanisms that will thwart each class of relevant attack. Plus, the architect will understand both the effects of system attack on the system under attack and the potential repercussions to the organization.

Still—this is important, and we will delve into this in some detail—a security architect will typically not be required to hold the details of every variation of each class of attack, and not required to command the specific technical details by which a particular instance of a class of attacks will exercise a particular vulnerability. This level of detail is often irrelevant to implementing a defense.

This isn't to suggest that security architects cannot master the details of a particular attack; usually, they will have to have explored at least one example of each type of attack in order to understand its mechanism sufficiently. And, of course, people move around. There are plenty of security architects who were once penetration testers and penetration testers who are competent security architects. The point is, the exact assembler language to exploit each of a collection of particular heap overflows is not required for the memory be handled in a secure fashion by a programmer. The literature and other training is full of examples of how memory is to be handled correctly.

Importantly for this book's tips and tricks to make as much sense as possible, Chapter 3 (Architecture, Attacks, and Defences) will explore the required attack knowledge set in some detail—that is, why to use attacks and how to derive a set of mitigations and defenses.

Chapter 4, Culture Hacking, is a tour of the approaches, tricks, and yes, even a few manipulations that have proven successful for practicing security architecture in the face of the sorts of challenges most will meet. I also include bits about starting, maturing, and running effective security architecture programs. At least, these things have worked for me and quite a few of the people with whom I've worked. Your mileage, of course, may vary.

In Chapter 5 (Learning the Trade), the secrets of the trade revealed herein will be set out in a series of short snippets, each hopefully delivering a bit of wisdom that can be applied as you, the reader, practice security architecture.

Finally, in Chapter 6, I've tried to set down, as best that I can, a lot of the tricks that I've used to surmount typical problems. Lucky for me, I get to interact with a lot of practicing security architects. My network has provided me with strong anectodal evidence that most programs will encounter many, if not all, of the challenges that I list in Chapter 6 as they mature. While not at all scientific, I've become convinced that there are milestone problems that crop up for nearly every program. I hope that it would be helpful if I listed these and then provided my methods for tackling them, for whatever worth you may get from these.

There are a few previously published pieces in the Appendices following the chapters that have been referenced during the text. I provide these to fill in any questions you may have, and for reference.

It is my sincere hope that *Secrets of a Cyber Security Architect* provides you a fun read, some insight, and also that it's organized well enough to be a desk reference as you proceed through your security architecture career and practice.

— Brook Schoenfield
August, 2019

Acknowledgments

Hopefully, *Secrets of a Cyber Security Architect* is *not* a work of fiction. The people mentioned herein by name are most assuredly actual people, though a few are sadly no longer living. This book concerns itself with practices that work versus practices that have proven counterproductive. The ideas, as Adam Shostack so wisely quips, "are abstract." As such, this is a human work, perhaps all to human? Unavoidably, the book is as much about people as technical matters.

There are, indeed, a number of fairly technical discussions here, as may be expected. Still, a great deal of the book is devoted to how people enact the abstract, how we move abstractions expressed through the practice of security architecture from ideas to working software that performs useful human functions and, at the same time, exhibits protective properties. With any luck, this book will help avoid adversary exploitation of software weaknesses that result in harm to the users and owners of the software. Make no mistake, adversaries are humans, too: adaptive, creative humans. Which brings us back to the very humanness that must underlie security architecture.

As I noted in the dedication to this work, hundreds of people at all skill levels contributed to this work. Without those who've tolerated me enough to try to learn from me, who've humored the many blind alleys that have eventually led to the teaching methods I employ today, this book would most certainly not exist. Whatever success my current methods have is due in large part to participants' patience, endurance, and persistence.

I arrived at Cisco in 2000 an extensively experienced software programmer, designer, and technical lead. Unfortunately, I was a rather less skilled security practitioner than I had believed about myself before I met people who were steeping themselves in cyber security. Thanks to Gil Daudistil and Doug Dexter for extending both their professional support and personal friendship, so that I survived that first year on Infosec's architecture team. The WebArch team, Steve Acheson, Laura Lindsey, Catherine Blackader Nelson, and Rakesh Bharania, may have hazed me a bit, yes. But they also pushed me to extend my skills. All remain treasured friends.

The Cisco Infosec identity management working group, circa 2001–2002, lead by Michele Guel, and including Steve Acheson, Steve Wright, Sergei Rousakov, and myself, grappled with what was at that time a huge paradigm shift: identity as an organization's "security perimeter."[*]

[*] At that time, cyber security defense focused on network protections.

It was through that effort that I began to find my feet as a security architect. We produced formative results that are just now gaining recognition close to 20 years later.

Having left the safety of a small software company to jump into the roiling waters of Big IT and Big Tech at Cisco, I was completely unprepared for the large number of highly skilled, highly motivated, high-integrity people with whom I've had the privilege to interact for the last 20 odd years. There is not space to name them all. That so many of my co-workers would become dear friends is the unexpected bonus of our shared passion to make people's digital lives just a wee bit safer. Or as John Stewart quipped one time when we were waiting for a plane to a conference at which we were both speaking, "Simply make the Internet work." Word!

John wrote one of the most lyric and beautiful Forwards I've ever read for my last book, *Securing Systems.* Thanks for your leadership. You guided what had been a dysfunctional organization to excellence. Thanks for your friendship. Obviously, thanks for being willing to stick your name onto my book. And, thanks for a few rides to the train, too. Those were important conversations, I think? At least, I carry them with me.

Without managers who see the value in playing a "long game," who've consistently seen the value of building a team first before focusing on delivery, much of what's in this book would not ever have been tried. Rob Rolfson, Michelle Koblas, Nasrin Rezai, the incomparable Dr. James Ransome, Scott "Chopper" Walker, Steve Mori.

Thanks are due to the first WebArch team that I led wherein we tried out the foundational concepts that through refinement blazed a trail to the material in this book. Vinay Bansal, Justin Tang, Ove Hanson each need to be acknowledged. Plus, there were a few project managers in that period who organized and supported the work: Caroline Thrasher, Ferris Jabri, Julian Soriano, and Dan Burke. Aaron Sierra helped me validate that the techniques also work for product security and cloud architectures.

The threat modeling exercises from which conclusions and suggestions herein are drawn have been greatly improved in collaboration with Damilare Fagbemi, who ran around Ireland and the United States co-teaching, and then incorporating class feedback into the class. A couple of other contributors to the class are Luis Servin, David Wheeler, Sun Lee, and Tania Skinner (though she may not realize the importance of her critique and validation).

James Ransome and I spent a couple of months in front of a whiteboard refining what we knew about software security and identifying that which we didn't. The proven ideas were then captured in *Core Software Security,* Chapter 9. Since publication of that book, we each keep honing those seminal concepts as well as finding new ones, a few of which are presented in this book, and most of which will be in our next.

No technical book should complete without technical review. The ideas presented here have received significant review from peers and then have been proven through practice; much of the practicing must be credited to the security architecture leaders of the programs I've led. There are too many peers to name: you know who you are.

Some of this work has been previously published in various forms: books, booklets, presentations, papers, posts. Along the way, specific technical reviews were provided by Jack Jones (risk), Izar Tarandach (security architecture and threat modeling), Blake E. Strom (unorthodox use of ATT&CK), Cedric Cochin and Raj Samani (posts covering various discreet subjects), Jon King and Celeste Fralick (continuing inspiration).

This book has been immeasurably improved through suggestions from Adam Shostack. Thanks to Adam for contributing his Forward, as well.

Thank you to the MITRE® Corporation for allowing me to reprint several of their copyrighted materials.

No book can move forward without the support of the publisher and publishing team: John Wyzalek for confidence in the concept. I apologize that the first draft took so long: sometimes the book I set out to write is not the book that emerges through my process. Thanks are due to Theron Shreve and his staff at Derryfield Publishing, with special acknowledgment for the diligence of copyeditor and typesetter Susan Culligan, who also provided significant project management.

Finally, but no less important is my daughter, Allison, who follows in Da's footsteps as a security architect. Dinner conversations invariably descend into technical discussions. She then continues to further prove these ideas through her work. My spouse, Cynthia, must lastly be mentioned. For one thing, she puts up with these technical discussions at the dining table. "It's the alien-speak, again." You have, with this one, suffered through the pain of writing five books about computer security. Without your humor, insight, and forbearance, none of this would unfold. Thank you.

About the Author

Brook S. E. Schoenfield is the author of *Securing Systems: Applied Security Architecture and Threat Models*[*] and Chapter 9: Applying the SDL Framework to the Real World, in *Core Software Security: Security at the Source*.[†] He has been published by CRC Press, SANS Institute, Cisco, SAFECode, and the IEEE. Occasionally, he even posts to his security architecture blog, brookschoenfield.com.

He is the Master Security Architect at a global cyber security consultancy, where he leads the company's secure design services. He has held security architecture leadership positions at high-tech enterprises for nearly 20 years, at which he has trained and coached hundreds of people in their journey to becoming security architects. Several thousand people have taken his participatory threat modeling classes.

Brook has presented and taught at conferences such as RSA, BSIMM, OWASP, and SANS What Works Summits on subjects within security architecture, including threat models, DevOps security, information security risk, and other aspects of secure design and software security.

Brook lives in Montana's Bitterroot Mountains. When he's not thinking about, practicing, writing about, and speaking on secure design and software security, he can be found telemark skiing, hiking, and fly fishing in his beloved mountains, exploring new cooking techniques, or playing various genres of guitar—from jazz to percussive fingerstyle.

[*] Schoenfield, 2015.

[†] Schoenfield, 2014.

Chapter 1

The Context of Security Architecture

1.1 Omnipresent Cyber War

At the time of this writing, those of us participating in the digital (online) world are living in an unprecedented age. Never before in human history have machines and the energy used to drive them performed so much of the labor required to sustain human life.

Consider for a moment mechanizations used for farming: tractors that pull tillers that can prepare hectares of soil for planting. Compare that to an 18th-century farmer who would hitch a draft animal to a plow, turning a single furrow at a time. Now even seed can be dispersed from a device pulled by the tractor.

What was once strictly the domain of human labor, perhaps including animal strength, can now be accomplished with the right equipment by one determined person.

That is not to say that farming cannot be accomplished by human labor alone, nor to say that it isn't by the many subsistence farmers still active on Planet Earth; certainly, people continue to farm with a stick or utilize animal power, if available and affordable.

I make this digression simply to illustrate the profound changes that have occurred over the last few hundred years in nearly every domain of human production, including information processing. The Computer Age (also called the Information Age and the Digital Age) has blossomed to magnify technology's reach and influence, for a computer can control a formally manual manufacturing process.

Lots of skilled and unskilled jobs have been replaced, and more are on the block. Perhaps truck/lorry drivers might be next? Anheuser-Busch has already delivered a load of beer with an automated truck.[*]

[*] Isaac, 2016.

In the context of farming, imagine that many (perhaps most?) farmers on the planet embed cheap moisture sensors in their soil, which they could monitor from their mobile phone. Since cell towers have long since leap-frogged land-line infrastructures in the Third World, there are mobile services available to many fairly remote areas. It is conceivable that even relatively poor people may have mobile phones in the near future.[*] This isn't idle speculation; I had this precise conversation with a principal of The Climate Corporation in 2015.

Computers are being adapted to either perform or assist with tasks, both new and old, at a dizzying and increasing rate; computers are everywhere surrounding us. Those of us living in the connected world are surrounded by an aura of radio frequencies the like of which has never been experienced, so far as we know. This bombardment with radio waves at many frequencies at the same time constitutes an unfolding experiment in human (and our animal companions') tolerance for radio transmissions. [Time will tell if we can actually tolerate the bombardment or what the effects may be. Perhaps evolution will step in with a genetic mutation for radio wave tolerance?]

I believe that we've really only scratched the surface in this transformation.[†] It is likely that computation capabilities will increase dramatically (quantum computation?) while at the same time size and power requirements will drop. Sensors of nearly everything sensible by computers will surround us, just as some futurists have imagined.[‡] It is quite likely that computers will start to write computer programs—at least, those parts of programs that are sufficiently deterministic and formulaic to be programmed algorithmically or heuristically (at least to start). [It is certainly within the realm of possibility that machine learning or artificial intelligence might be applied to the generation of software algorithms.]

I can imagine a digital world far more complex and rich than exists today, which is far beyond what I imagined possible 40 years ago, when I was introduced to programming. A great deal has changed. I think that far more is in front of us than behind. Our lives are incredibly dependent upon computers and the software that they run.

We also know that the software we depend upon is riddled with errors, whose correctness cannot be automatically proven (that is to say that at this moment, the Turing Proof still holds true). Which brings us squarely to the security problem those of us in the connected world must face, every day, sometimes many times a day: There be pirates on these digital waters, Matey.

> *Our ability to develop complex software vastly exceeds our ability to prove its correctness or test it satisfactorily within reasonable fiscal constraints . . . complex software is difficult to write and to test, and will therefore contain numerous unintentional 'bugs'[.] It would be extremely difficult and expensive to determine with certainty that a piece of software is free of bugs[.] Given the relatively small amounts of funding allocated for developing and testing . . . software, we may safely consider it as effectively impossible.*[§]

A couple of quotes from the Forwords to *Securing Systems* may help to illustrate our digital dependence:

---

[*] Ogundeji, 2015.

[†] Osborne, 2018.

[‡] For instance, the nano digital world imagined in Neal Stephenson's *Diamond Age* (Stephenson, 1995).

[§] Rivest and Wack, n.d., pp. 3–4.

"We are struggling as a security industry now, and the need to be successful is higher than it has ever been in my twenty-five years in it. It is not good enough just to build something and try and secure it, it must be architected from the bottom up with security in it, by professionally trained and skilled security architects, checked and validated by regular assessments for weakness, and through a learning system that learns from today to inform tomorrow. We must succeed."[*]

"Virtually every aspect of global civilization now depends on interconnected cyber systems to operate. A good portion of the money that was spent on offensive and defensive capabilities during the Cold War is now being spent on cyber offense and defense. Unlike the Cold War, where only governments were involved, this cyber challenge requires defensive measures for commercial enterprises, small businesses, NGOs, and individuals."[†]

Failure to understand that the dependence that we have not just on our obvious digital devices—smart phone, laptop, tablet, fancy fitness bling on your wrist—but also on a matrix of interconnection tying all these devices and billions more together will land you in the hot seat; consider what happened to THE HOME DEPOT®, whose management actively resisted understanding their business's interconnectivity.[‡] When THE HOME DEPOT's security folk pointed out the glaring weaknesses in the company's cyber defenses, an executive is quoted as declaring, "we sell hammers, not computers." The sad truth is that even a hammer and nail company must still account for its digital operations, which nearly always imply maintaining a cyber security defense posture. The breach occurred in 2014. By 2017, that breach had cost the company $179 million.[§] Not exactly chump change, even for a global corporation.

For more than three billion out of the seven billion people on this planet, we have long since passed the point at which we are isolated entities who act alone and in some measure of unconnected global anonymity. For most of us, our lives depend not just upon technology itself, but also on the capabilities of innumerable, faceless business entities and those entities' digital systems that act upon our digital behalf.

Consider the following common, but trivial, example: When I swipe my credit card at the pump to purchase petrol, that transaction passes through any number of computation devices and applications operated by a chain of business entities. The following is a typical scenario (an example flow—but not the only one, of course):

- The point-of-sale device itself (likely supplied by a point-of-sale provider) (see inset)
- The networking equipment at the petrol station (see inset next page)

My friend and former colleague Lucy McCoy wrote the communications code in the first generation of gas pump payment terminals. At that time, terminals communicated via modem and phone line. She was a serial communications wiz. I remember the point-of-sale terminal laid out in her lab area. Lucy has since passed away. She was a brilliant engineer; she gave my code the best testing that any code I've written has ever received.

[*] John N. Stewart SVP, Chief Security & Trust Officer Cisco Systems, Inc.
[†] Dr. James F. Ransome, CISSP, CISM.
[‡] Please see brookschoenfield.com/?p=219 for more discussion on THE HOME DEPOT's large breach.
[§] WebTitan, 2017.

- The station's Internet provider's equipment (networking, security, applications—you have no idea!)
- One or more telecom company's networking infrastructure across the Internet backbone
- The point-of-sale company or their proxy
- More networking equipment and Internet providers
- A credit card payment processor
- The card issuer who must validate the card and agree to pay the transaction for me

And so on . . . all just to fill my fuel tank. It's seamless and invisible—the communications between entities usually bring up an encrypted tunnel, although the protection offered is not as solid as you may hope; it is invisible and seamless, except when the processing is not so invisible, such as during a compromise and breach. [On a trip through Idaho several years ago, a fuel pump in a remote station from which I fueled had a card skimmer attached to it. My credit card was then used for fraudulent activity.]

> The transactions have to get from station to payment processing, right? Who runs those cable modems and routers at the station? Could be the Internet provider, or maybe not. I run my own modem/routers/switches at home to which I have sole admin access. An em--ployee might easily slip a router under their control between terminals and provider; who's to know? You don't walk up to the payment kiosk and demand to see the routing gear, do you?

Every one of these invisible players has to have good enough security to protect me, and you, if you also use some sort of payment card for your petrol.[*]

The foregoing is one of many examples:

- Medical gear and the networks that support them.
- Your financial institution's systems (where your "deposits" are really just digital data).
- The 100–200 processors in your car, with the stack of software that runs on them.
- Your Bluetooth headset—a Linux computer, as is that webcam watching your front door.
- Your smart TV—another Linux computer, as is your printer, the thermostat, Alexa, Echo; even your landline's handset may very well be another Linux system.

Is there nothing sacred? Probably not. And all of it is attack surface if there's anything worth stealing, learning, coercing, influencing, misusing, or disrupting to be found. Such has become the nature of the Connected World: it's all attackable given sufficient motivation.

The art of security architecture plays a part in the dance between adversaries and defenses. The security architect attempts to align defenses to expected attacks. As we shall see in Chapter 3, Attacks and Defenses, the security architect seeks to understand relevant attackers, their methods, and their goals. At the same time, they must also understand how particular types of attacks work such that appropriate defense mechanisms can be specified, implemented, and deployed.

Security architects may be thought of as software and system architects who specialize in attacks and defenses, who are proficient enough with architecture techniques to specify defenses as part of the structure of a system or software architecture.

I don't believe that our present and future digital security rests solely on this one discipline; that would be incredibly arrogant and also quite unfair to the many other disciplines that have emerged within the digital security space, such as:

[*] http://brookschoenfield.com/?p=219

- Exploit and vulnerability research
- Analysis of the dynamics of the threat landscape and human adversaries ("threat research")
- Malware analysis
- Incident response
- Risk analysis
- Defensive programming
- Vulnerability and error discovery tools and automation
- Constructing easier-to-secure programming languages and environments
- The defensive software industry
- Management of the problem space's complexity
- And so forth

Still, as was noted in NIST 800-14 in 1996,[*] if we cannot uncover appropriate security requirements during system design cycles, we have already lost an important—nay, key—opportunity. We leave ourselves with the difficult (and usually far more expensive) challenge to amend insecure software late in the game, perhaps even after deployment. That is the essence of this book: to add to a growing body of practice and, I hope, wisdom about what security architecture is, why it's important, and how to practice it successfully.

In the following chapters, sections, and pages, I hope by collecting some of the bits and pieces that I've found useful into one volume, I can contribute in some small way to the art and science, the practice of this thing that's become known as "security architecture," to which I've given some of the best of my professional life and quite a bit of my thinking.

Some of the following has been pulled from other works of mine. To that material, I've added additional thoughts and learnings derived since those publications. I've also tried to augment ideas that I've touched on in passing with greater depth. Let me know what you find useful, as always.

1.2 Know the Threat Actors

Attack and the subsequent "compromise"—that is, complete control of a system on the Internet—is utterly pervasive: constant and continual. And this has been true for quite a long time. Many attackers are intelligent and adaptive. If defenses improve, attackers will change their tactics to meet the new challenge. At the same time, attack methods that were once complex and technically challenging are routinely "weaponized": turned into point-and-click tools that the relatively technically unsophisticated can easily use.[†] This development has exponentially expanded the number of attackers. The result is a broad range of attackers, some highly ingenious alongside the many who can and will exploit well-known vulnerabilities if these are left without remediation.

The chance of an attempted attack of one kind or another is certain. The probability of a web attack is 100 percent; systems are being attacked and will be attacked regularly and continually. Most of those attacks will be "doorknob rattling"—reconnaissance probes and well-known,

[*] NIST, 1996.

[†] One recent example of such weaponization is SpookFlare 2.0, available from public GitHub servers for download at https://github.com/hlldz/SpookFlaref6e0>

Fifty million is a number given to me by intrusion analysts at a major high-tech company. At these levels, it doesn't really matter if the number is more like 30 million or even 100 million. The number of attacks overwhelm even the best staff unless they have significant automation.

easily defended exploit methods. But out of the fifty million attacks each week that most major websites must endure (see inset), something like one or two within the mountain of attack events will likely be highly sophisticated and tightly targeted at that particular set of systems. And the probability of a targeted attack goes up exponentially when the web systems employ well-known operating systems and execution environments.

A web server listening for connections from the Internet must remain open to all Internet traffic [unless IP address restrictions are put into place such that only some networks or even particular hosts can access the web server], which means that any attacker can probe the interface, at least at the network protocol level, if not deeper. The constant doorknob-rattling sweeps of the Internet will surely find and investigate whatever interfaces are open and available to unrestricted traffic.

Once an interested attacker finds and catalogs the open HTTP port, then the fun really begins. Like web vulnerability scanners, the attacker will probe every reachable page with every attack variation possible. These probes (and possibly downloads of portions of the site) will be unrelenting. To prevent restriction of the attacker's address, they may use multiple addresses or even shift attacking addresses on the fly (e.g., fast flux DNS).

In contrast, security architects must use their understanding of the currently active threat agents and their techniques in order to apply these appropriately to a particular system. Whether a particular threat agent will aim at a particular system is as much a matter of understanding, knowledge, and experience as it is cold hard fact. Considering the potential effects of threat agents and their capabilities to attack any particular system is an essential activity within the art of threat modeling. Hence, a security assessment of an architecture is an act of craft that wields engineering as the tool set.

Although in practice the order in which we consider different aspects of attacks and defenses doesn't materially affect the quality of the output, let's start with threat agents (actors, adversaries). The goals and capabilities of each *type* of actor profoundly affect how deep and thorough a defense must be. I explained this in *Securing Systems* at some length: some actors seek a quick return on effort. Others will work on a compromise until successful, no matter the time and cost. Some adversaries mean to cause harm, some mean to cause no harm, and some don't care about what harm may ensue, so long as a goal is achieved.

There are other dimensions to consider. I've settled on five areas:

1. Final goal of the attacker
2. Technical ability
3. Risk tolerance
4. Work factor
5. Activity level

There's nothing sacred about my categories. These threat actor attributes are the ones that allow me to set priorities when analyzing a system. If you don't like these or see holes in my thinking, then by all means use your own categories.

After years of refining the matrix, years of teaching threat modeling to diverse groups of people (from groups of a few individuals to 120 participants), and several years of having participants in my sessions build their own matrices, this set of categories seems to work fairly well. Still, I remain open to other approaches. There is probably a better way; if you find a system that works better for you, please let me know.

Below, I briefly explain each of these threat actor attributes or behavioral dimensions:

- **Goal.** Many successful compromises depend on the successful execution of a set of exploits, one after the other. From reconnaissance, to establishment of a presence, to privilege escalation, on through establishment of command-and-control and persistence, none of these is typically the ultimate goal of an adversary [except for vulnerability hunters and security researchers, who may be satisfied with establishing proof that an exploitable condition exists in isolation]. For real-world attackers, the adversary is after *something*; perhaps stealing credentials or controlling the host so that it may be employed as part of a botnet, stealing information, disrupting operations, or just plain and simple theft of money or other assets—attackers have many goals. In order to identify which assets may be valuable to the set of attackers who are likely adversaries, it helps to understand what results different classes of attackers expect to achieve.
- **Technical ability.** Not every class of attacker wishes to employ highly sophisticated, resource-intensive techniques. For a moment, ignore the potential for any attacker to be highly sophisticated and to have access to powerful compute resources; there are classes of attackers who, even if they had such resources at their disposal, might not bring them to bear simply because it's too expensive. For this reason, it's useful to understand the sorts of technical capabilities, the sort of exploits, that a particular class of adversary is likely to use.
- **Risk tolerance.** What I mean by this attribute is how willing or unwilling a particular attacker may be to getting caught or to having the attack discovered and when. Spies tend to be highly secretive. The best outcome is if the activity is never uncovered or, if discovered, that it is difficult to attribute to any particular group or state, and the goal of the action should certainly be obscured as much as possible. Cyber criminals without a doubt know that once your account has been drained, you will notice. The thief doesn't care that you find out, only that the thief doesn't get caught—or at least, only low-level participants in a crime organization get caught; the upper level management must be protected, usually at all costs. On the other hand, security researchers are generally not breaking any laws, so they expect to publish the results with no risk. Security researchers may then be thought of as having zero risk tolerance.
- **Work factor.** This attribute is an estimation of how hard a particular class of adversaries will work toward achieving their goal. For instance, the United States National Security Agency had, at one time, a $60 billion so-called "black budget." That seems to me to be an enormous amount of resources to apply to any particular cyber action. I assume that other major powers have similar budgets. Contrast this with a cybercrime business that needs to maximize the amount of profit from each operation. As I noted in *Securing Systems,* cyber criminals tend to focus on the poorly defended, spending as little on research and development of new techniques as possible. This is a vastly different approach than would be taken by a well-funded superpower's offensive cyber activities.

The foregoing categorization is entirely stereotypic. It must be understood that an individual actor can be an outlier; it is important to know that aggregating behaviors as I'm doing implies beyond any doubt that some threat agents in each group will not fit the profile in one manner or another.

The advantage of stereotyping and aggregating is to allow us to step away from a widely held fallacy of practice: that every actor is either a creative, innovating genius or a so-called "script kiddie" of no significant technical capability. There is a continuum of capabilities and clusters of attributes into which particular actors tend to fall. Even technically sophisticated actors may have very good reasons for using readily available, well-known, and understood exploits: new vulnerability discovery and subsequent exploit development for that vulnerability is expensive. When trying to maximize profit, businesspeople try to minimize research and development. As a profit-seeking business, cybercrime is no different from any other profit-seeking enterprise.

Along with the fallacy of either unlimited technical capability or zero knowledge comes another: it is widely believed that every vulnerability/exploit pair is equally valuable to every actor. As I demonstrated in a series of analyses published on McAfee's Securing Tomorrow blog in November 2017,[*] vulnerabilities first must offer an attacker something of attacker-value that will advance toward the attacker's ultimate goal.

This is not a hypothetical example. I have entered into discussions many times with security professionals who didn't seem to understand the underlying lack of attacker value of additional memory manipulation after an attacker has gained full control of an operating system.

My classic example is a buffer overflow requiring high privileged access before it can be exercised (see inset). A buffer overflow allows the attacker to execute code of the attacker's choosing. However, having obtained high system privileges, attackers can already execute code of their choosing. They have no need to exploit yet another condition. With high privileges, not only can code be executed, but the attacker has access to all of the monitoring ....and logging capabilities on the system, such that their code can be hidden (with whatever facilities are available) and persisted through restarts of the operating system. There is no value to further exploitation of memory; the attacker already owns all memory on the system.

Once an attacker can insert her/him/themself into the kernel, it's "game over." The attacker has the run of the system to perform whatever actions and achieve whatever goals are intended by the attack. For system takeover, the kernel is the target. The highest operating system privileges typically allow access to the kernel. Hence, high privileges generally mean that the system's kernel is "owned"—or in the parlance, "pwned": compromised, under the control of the attacker.

The foregoing is basic risk analysis that every security practitioner ought to be able to perform. Still, I have seen precious few methodologies for including attacker value in the risk analysis equation. That's why I like the term "impact" vs. "loss." Impact is broader, such that it includes the possibility that an attacker has moved one step closer to their goal rather than focusing solely on the harm if the attacker should ultimately be successful.

Neither of these fallacies serve defense well, because they are both demonstrably untrue and create a situation (or mindset, unfortunately rather widely held) that implies that all issues must be defended against equally well (or worse, "fixed immediately"!).

[*] Schoenfield and Quiroa, 2017.

Because few organizations and few systems can be successfully built and maintained to defend against every possible and imagined attack, the practitioner is thus rendered ineffective as a direct result of the two misconceptions expressed above. That is, smart engineers will see through "all or nothing" arguments as technically invalid and impossible. Smart engineers want to know where to place the most efficacious defenses; they want to have solid reasons for implementation *before* they will agree to implement anything. When a security practitioner insists upon defenses that seem unreasonable to those who must implement them, then security's influence has been squandered. I have seen it over and over again; once developers no longer trust security, they will evade security requirements as often as they can.

Please don't mistake my meaning. It is possible to make mistakes and then to admit error. In fact, doing so builds the required trust between security and developers. That is very different from insisting on defenses that have little chance of being tested by real attackers simply because it feels safer to a security practitioner or because some book or standard said that these defenses were "always required for every situation." As far as I know from my 20 years of security practice, there is no "always" in defenses; each defense is contextual.

It's just not true that every nation-state cyber army is gunning for every connected person. There are distinctions, and applying these distinctions allows us to focus on the most likely attacks, at the most likely levels of sophistication, from actors who have readily observable levels of risk tolerance and who will expend varying levels of effort to achieve goals. The astute defender usually needn't pay attention to everything, all the time—which is in any case an impossible charge (see inset).

If I pull my five threat actor attributes together into a set of four attack pre-conditions, I come up with the following:

- There must be active, motivated threat agents who are interested in attacking systems of the sort under assessment.
- The attack methods required must lie within the technical capabilities of the attacker and be well enough understood by the attacker to be useful.
- The attack must not expose the attacker to negative consequences beyond the attacker's tolerance for exposure.
- The effort needed to complete the attack must be less than the expected rewards of success.

Still, and nonetheless, if successful compromise by an attacker has no impact or loss to the owner of the system—the organization whose goals the system is intended to further—or the system's users, then there can be no risk. Without impact, there can be no risk to an organization, even in the face of an easy-to-exploit condition.

In the foregoing discussion, three often overlooked attributes of an attack emerge:

- Impact
- Attacker value
- Required effort

Except perhaps defenders whose adversaries include dedicated nation-state actors. For this select set, the defender must implement as much as possible, knowing that every defense will likely eventually fail against the onslaught of highly resourced and sophisticated adversaries. The concept in this case is to slow the attacker down sufficiently that there may be time to catch the attack as it unfolds. In addition, each defense should add detail to the emerging picture of the attack, even as it may fail.

The first two, impact and attacker value, are not necessarily equivalent, although sometimes that which an attacker intends is, in fact, the impact to the victim: bank accounts drained, services disrupted. Sometimes, though, the relationship between the value of an exploit and the attacker's ultimate goal is more tenuous.

Consider the formation of the Mirai camera botnet[*] that was used to disrupt Internet communications through a DNS service distributed denial of service (DDOS) attack. Compromise of each Mirai camera quite likely was not even noticed by the camera's owner. The camera appeared to be functioning in whatever capacity for which it was installed.

Although the controller of the botnet (the "attacker") could capture images from cameras and may very well have, there is no indication that owners of the cameras were particularly inconvenienced, even during the DDOS attack.

The victim of the attack seems to have been a company called DYN, whose DNS services were disrupted. Alongside DYN were some users of a few major Internet services in select regions. Very likely, some of those who couldn't access FaceBook® during the attack owned one or more of the cameras causing the attack.

Still, the loss of trust in DYN's services and lost advertising revenues to Facebook are the organizational costs, while inconvenience from prevention of use of services would be another impact. These are quite disconnected from compromise of tens of thousands of Mirai's underlying Linux® operating system—which was the attacker value: controlling thousands of Internet-connected Linux devices.

To focus on either impact or attacker value in isolation without considering the other attributes paints an incomplete picture. We must understand whether an attack scenario will help advance the attacker's goals. And, we also must consider whether attacker success will affect something that system stakeholders care about protecting.

The third item above, "required effort," can often help us to understand the attacker's work/reward ratio. How much is a successful attack going to deliver for the effort?

For exploits that require a great deal of setup and perhaps are only a stepping stone, the reward has to justify the entire set of actions. Plus, if any of the preconditions are generally unknown to attackers, they will need time to discover these for themselves (unless all the necessary steps and preconditions get published in a research paper).

In the Mirai camera case, the effort was next to nothing: the default password was well known. The password was used via a well documented protocol (SSH) to access the camera at high privilege. Setting up the botnet was only a matter of discovering eligible targets [At which, it must be said, attackers, professional bug hunters, penetration testers, and security researchers are all very good. These all use an intersecting and constantly improving suite of tools, tools which are readily available, often open source].

In contrast, the WiFi authentication attacks named "KRACK Attacks" (<u>K</u>ey <u>R</u>einstallation <u>A</u>tta<u>ck</u>s) discovered by Mathy Vanhoef, published in October of 2017,[†] require fairly deep understanding of the intricacies of WiFi's WPA2 key interchanges. As I wrote in my blog just after publication of the new technique:

"However key reinstallation depends on either working with the inherent timing of a Wi-Fi during a discreet, somewhat rare (in computer terms) exchange or the technique depends upon

[*] Fruhlinger, 2018.

[†] Vanhoef, 2017.

the attacker forcing the vulnerable exchange through some method (see below for examples). Both of these scenarios take a bit of attacker effort, perhaps more effort than using any one of the existing methods for attacking users over Wi-Fi?"[*]

Whenever a new attack's value overlays existing, well-known, perhaps already weaponized efforts, then the attack's effort versus gain equation becomes important. Is the new attack method easier or harder? Is it more easily coded into a repeatable, automated form ("weaponized")? Is it harder or easier to detect? Does the new method extend in some way the reach of the attacker?

Attacker effort to pull off a successful KRACK seems to have been greater than the use of any number of other, readily available methods. So far (up to this writing), KRACK doesn't seem to have received much threat actor usage, as interesting as it may be from a computer science and security perspective. Of course, such research can lead to much tighter security implementations, which is precisely what has happened in this case; my thanks to Mr. Vanhoef for excellent research.

The foregoing examples, I hope, help to clarify dimensions that can be used to understand attackers and how they work, as well as to analyze and qualify issues as they arrive—these activities, I believe lie within the expertise that security architects must master.

As we consider different threat agents, their typical methods, and most importantly the goals of their attacks, I hope that you'll see that some attacks are irrelevant against some systems: these attacks are simply not worth consideration. The idea is to filter out the noise such that the truly relevant, the importantly dangerous, get more attention than anything else.

An astute reader might question why I have proposed yet another threat system?

The answer is simple: extant methods such as the Diamond Method,[†] MITRE's CRITS,[‡] and the UCO ontology,[§] to name a few, are focused on the problem of analyzing artifacts of an attack or an attack campaign.

These existing approaches are useful, certainly; I encourage readers to follow the references. There's a great deal to be learned from the study that's been put into helping analysts figure out what threat actors are doing and why. An attack might be a singleton, perhaps highly targeted, or it might be one part of a campaign against numerous targets. When reacting to a potential attack, and for researchers who are trying to attribute attacks to particular actors, graphing malware samples, targets, period of attack, originations, etc. (please see any of the references, above) will be critical.

But for security architecture, it is sufficient, I believe, to know that such attacks occur and to generalize about what different classes of attackers seek. That is the knowledge set that is wielded to assess the *potential* for successful compromise:

- What does the attacker ultimately want to achieve?
- What are the attacker's methods, both known and probable?
- How are the attacker's methods applied to vulnerabilities, both known and potential?

[*] http://brookschoenfield.com/?tag=kracks-attacks
[†] http://www.activeresponse.org/wp-content/uploads/2013/07/diamond.pdf
[‡] https://crits.github.io
[§] https://github.com/ucoProject/UCO

- How will a successful attack affect assets, users, owners, and ultimately the organization that must be protected?
- What steps can be taken to thwart attack? That is, what are the best defensive measures against likely attacks?

These are the questions that need to be answered by security architects. We have to be conversant with those attacks that will be levied against systems under analysis, and the sorts of defenses that have been effective against such attacks. This is a set of knowledge that overlaps something like MItre's CRITS, but which must do two different things:

1. Attacks must be grouped into methods that obtain attacker intermediate or ultimate goals and their targets.
2. The security architect makes an educated guess, as highly informed a guess as possible, about what might happen; security architecture is meant to be proactive—before attack, not during or after.

There is guesswork involved—hopefully before entering the real world of probing for cyber weakness—that is, before attackers begin to probe the system under analysis. Plus, threat modeling will be improved markedly through a feedback loop between the informed guesses used during the threat modeling analysis and validation (or not) of the guesses by penetration testing (see inset).

Threat modeling and penetration testing might be considered bookend techniques for a robust and mature software security practice. Threat modeling (really, any and all architecture analysis for security, whatever it may be called, started at an early stage of structural conception) is an up-front analysis meant to strongly influence the structure (architecture) and design of systems as they are being built and implemented. Penetration testing "proves" the security posture that was intended to be built.

Can you perhaps see how these two activities would influence each other? Penetration testers, at the very least, must understand the visible and accessible architecture of a system in order to probe its defenses and identify its weaknesses. If weaknesses are found, then the threat model must be updated to account for any missing defenses that then can be added.

> These two processes—threat modeling and penetration testing—are too often conducted by separate entities who are not working in concert. It was Eoin Carroll, in his capacity as a Senior Security Architect at McAfee, who helped me to understand the importance of tying these activities tightly together. Threat models can and should be proved through testing of many sorts. Please see Chapter 9 in *Core Software Security* for more about the use of various software security testing techniques.

Penetration testers act as a proxy for real-world adversaries. Hopefully, before the system is exposed to adversaries, penetration testers can find any holes in the required defenses so that those holes can be plugged enough to sufficiently resist attacks by those who intend to damage the system and/or its owners.

Threat modelers—that is, security architects—must understand attackers, their techniques, and their goals (both short-term and ultimate). These so-called "guesses" must be made sufficiently accurately such that defenses will be built as the system is being built. This early analysis is an attempt to avoid so-called "bolt-on" additions of security defenses after implementation, or worse, after go-live, or worse still, after compromise. "Built-in" is much cheaper than bolt-on. Built-in will presumably fit the overall plan of the system (its architecture).

Hence, the foregoing is the purpose of my high-level tables; they provide a quick reference frame for making stronger guesses.

Table 1.1 Summarized Threat Attributes

Threat Agent	Goals	Risk Tolerance	Work Factor	Methods
Cyber criminals	Financial	Low	Low to medium	Known proven
Industrial spies	Information and disruption	Low	High to extreme	Sophisticated and unique
Hacktivists	Information, disruption, and media attention	Medium to high	Low to medium	System administration errors and social engineering

Table 1.1, copied from *Securing Systems*, and its completion in Table 1.2 are meant to assist in the previously described process of analytical educated guessing (see inset). Security architecture should be applied before the system is completely implemented, if possible, while there is still room for change (to account for new defenses). Thus, the analyst is, in my humble opinion, greatly aided by bearing in mind just who will attack, why, and what techniques they are likely to employ in their efforts. Since these are (highly) educated *guesses,* it is, again, in my humble experience, not necessary to fully understand each attacker's strengths and weaknesses in detail; a stereotypic picture of *types* of attackers is generally all that is required, such as in the tables presented herein.

Penetration testing against a complete or nearly complete system can then test the security architect's guesses, as my friend and colleague Eoin Carroll likes to say. Eoin has shown me that there's a natural feedback loop between a security architect's threat model and a subsequent penetration test. The penetration test can prove that the threat model has been thorough enough. Or, if it has not, whatever penetrations (that is, successful attack tests) have occurred must improve the threat model.

In Securing Systems, I dive deeply into the threat modeling process, hopefully putting flesh onto the bare bones laid out in this book. Readers are encouraged to peruse any of the six full system analyses contained in that work for more information.

In *Securing Systems,* I examined in some depth three types or classes of threat agents:

1. Cyber criminals
2. Industrial spies
3. Hacktivists

I did not complete the matrix, because *Securing Systems* was meant to be as much a series of exercises in threat modeling analysis as a definitive work on security architecture knowledge and patterns. I left the remainder of the matrix as an exercise for the reader.

Table 1.2 High-Level Threat Agent Attribute Matrix

Threat Agent	Goals	Technical Ability	Risk Tolerance	Work Factor	Activity Level
Cybercrime	Monetary	Low (known proven)	Low to medium	Low	Very high, continual
Industrial espionage	Information	Medium to medium-high	Low	Medium	Low. For enterprises, medium
Nation-states	Information disruption	Very high	Very low	Very high	Medium but constant
Law enforcement/government compliance	Compliance information	Medium	None—they are the law	Medium	Intermittent
Insider	Monetary	Varies	Low	None	Occasional
Insider	Revenge	Varies	Very high	None	Occasional
Usage abuse[a]	Unauthorized use	Low	Low	Low	Constant
Hacktivists	Media attention for cause	Low to medium	Used to be high, now much lower	Medium	Intermittent
Hackers	Status	Often very low	Low	Low	Low
Security Researcher	Career enhancement	High	None	High	Medium

[a] I'm indebted to my students who added this important adversary, users who seek to obtain more services than those for which they've contracted. Users are not in the same technical category as cyber criminals, though their acts may be criminal, depending upon the jurisdiction. They aren't generally making a living through their attempts to enhance their services. They often simply feel entitled to "more." Consider abusers seeking more content streaming than that for which they've paid.

As I began to incorporate some of the new material that I developed from Securing Systems into my classes {see inset next page], I brought the exercise of building a threat agent matrix as a part of the coursework. Very quickly, class participants requested a completed matrix.

In my classes, I hand out the matrix after each team has completed their very own threat agent attribute table for themselves, and after each team has presented their matrix to the other teams. The exercise seems to be quite powerful, in that it gets everyone in the class thinking about who

Damilare Fagbemi, Senior Security Architect at Intel, Inc., helped refine the third version of the threat modeling class that I continue to give. Damilare delivers a similar class for Intel. We regularly discuss our various teaching discoveries and challenges. Damilare's continuing support has been critical to whatever success my threat modeling classes achieve. His contributions remain vital, not just to our classes, but to the industry as a whole.

their adversaries are and what are their typical attributes. The very act of considering these problems sets up participants with the correct mindset to play adversary to systems under consideration.

Because the exercise has proven so effective, I'm quite hesitant to put a completed threat agent matrix into a published work. The matrix has to be a "living" document; the cyber adversary threat landscape is dynamic and constantly changing. Classes of actors and their typical attributes are going to change over time. In no way should Table 1.2 be considered canonical.

Plus, your experience with your adversaries may be different than mine. Please take Table 1.2 as a suggestion on how this problem might be approached, an attempt to build a high-level picture of types of threat actors, what each type's stereotypical goals are, and something about their capabilities and tolerances. Please do not take my threat agent matrix as gospel. Instead, treat the matrix as a pointer, as an exercise in assigning some order for a dynamic and somewhat opaque problem space. Make your own table; decide upon attributes that bring order to your threat landscape.

1.2.1 Useful Exploits Don't Die

In fact, it may be said that, "Old exploits don't die, they just fade away," with ever diminishing use. Unfortunately, as a defender, one can never toss an aging and seemingly forgotten exploit technique into the dustbin of history. Attackers will use anything at hand—old, new, whatever. Just because many target systems have been upgraded to plug a vulnerability doesn't imply that there aren't existing systems that are still vulnerable. For many attacker goals, a step forward may be achieved via a single, vulnerable system.

Importantly, not much seems to completely disappear from the Internet. Old machines, running long unsupported, end-of-life software get sold, donated, or given away. These systems still remain connected *somewhere*, and thus remain potential victims of exploits supposedly long past. The grim reality is that these systems and their pirated knockoffs make up some part of the complete Internet demographic, often being used in poorer areas and/or by less knowledgeable people.

It may be impossible for users' systems at the bottom of this chain of reuse to fix the existing vulnerabilities on their systems since much or even all of the software on the systems may be past support—long past. That is, the makers of the software no longer fix issues, even newly discovered issues. The owners of the systems are often just out of luck.

The nasty truth of this situation is that there always remains a vulnerable population against whom old exploits may successfully be employed. So, why should the builders of exploit kits ("EK" in security parlance) remove old exploits? There's no compelling reason to.

Thus, EK are *additive.* New exploits are added, but old ones are not removed. EK are the workhorses of attackers; some EK are maintained by the developers of the adversarial world. [Some EK are maintained by security researchers for the benefit of researchers and penetration testers. That does not prevent these EK from also being used by adversaries.] The developers may not use their kits against victims; their business model is as a support to actual attackers. Attackers well supplied with already programmed—that is, pre-canned, also called "weaponized"—exploit code are then freed up to concentrate on their attack campaigns rather than mucking about identifying code to exercise particular vulnerabilities.

Attackers work within their own rich ecosystem of researchers, developers, service offerings, consultants, etc., which mirrors the legal software business ecosystem. It's a strange and complex digital world that we live in.

1.3 Everything Can Become a Target

I've been declaring for years that, "whatever can be engineered by humans can be reverse engineered by humans." That is, in this context, whatever protections we build can ultimately, with enough resources, time, and effort, be undone (see inset). This is an essential piece of the probability puzzle when calculating or rating computer security risk. The fact that the attackers can learn, grow, and mature, and that they will rapidly shift tactics, indicates a level of heuristics to the defense of systems: expect the attacks to change, perhaps dramatically.

My assertion lies at the heart of modern cryptography. Cryptographic strength is measured in "work-years" to decrypt without the keying materials. The assumption being that there is no encryption that cannot be undone, given sufficient time and resources.

A (hopefully) informative example of the above truism might be taken from a pair of processor (CPU) issues named Spectre[*] and Meltdown.[†]

As Adam Shostack, author of *Threat Modeling: Designing for Security,*[‡] so succinctly put it: "The back and forth of design and critique is not only a critical part of how an individual design gets better, but fields in which such criticism is the norm advance faster."

The Spectre/Meltdown issues are the result of just such a design critique process as Shostack describes in the pithy quote given above (see inset).

In my humble experience, Adam is particularly good at expressing complex processes briefly and clearly—one of his many gifts as a technologist and leader in the security architecture space.

Let's look at some of the headlines from before the official issue announcement by the researchers:

The Register: Kernel-memory-leaking Intel processor design flaw forces Linux, Windows redesign.[§]

Wired Magazine: A Critical Intel Flaw Breaks Basic Security for Most Computers."[¶]

There were dozens of these headlines (many merely repeating the first few, especially, *The Register*'s), all declaiming a "flaw" in CPUs. I want to draw the reader's attention to the word "flaw." We shall dig into the applicability of that designation to these particular issues in order to highlight a constructive dialog that should occur around designs as our understanding of the design's security posture matures over time. *The Register* reporting was based largely upon speculation that had been occurring among the open source community supporting the Linux Kernel following a couple of changes that had been made to kernel code. It was clear that something was amiss, likely in relation to something in CPUs; concerned observers were guessing what the motivation for those code changes might be.

[*] Kocher, P., Genkin, D., Gruss, D. et al., 2018.

[†] Lipp, Schwarz, Gruss et al., 2018.

[‡] Shostack, 2014

[§] Leyden and Williams, 2018.

[¶] Greenberg, 2018.

It may help to step back just a moment from Spectre/Meltdown details to see how a report of issues is typically (though far from always) handled. One approach is known as "responsible disclosure." But not all researchers believe in responsible disclosure (more on that below). The supposedly "responsible" process involves embargoing (preventing from inadvertent revelation) the existence of the issues and their technical details until a fix can be readied by those responsible to produce fixes. That is, issues are to be kept secret until users can be protected.

If the facts of the issues' existence—or worse, the technical details—are made known ahead of the availability of a fix, then attackers have a terrible advantage. Attacks can proceed without even the hope of a direct defense. Hence, the "responsible" in "responsible disclosure": don't give attackers any help.

How anyone could argue with the logic of responsible disclosure is beyond me, frankly, though I've listened to arguments against it. These make no sense to me, except from what appears to be a self-righteous and/or completely disconnected perspective. "Do no harm" is one of my main aspirational dictums in life. However, there are those who believe otherwise, who believe in immediate, "zero day" disclosure, users' protections be damned.

There appears to me to be some sense that so-called "full disclosure" is the only weapon researchers have to somehow force software makers to deliver fixes. Having lived on the other side of that argument, I can assure my readers (at least) that any honest software maker (I wouldn't work with dishonesty) often has many business drivers with which to contend. Plus, not every fix is a few lines of code. If a major product needs to be redesigned, there isn't any way to deliver a "fix" immediately. Spectre and Meltdown actually fit into this case: they are artifacts of design decisions not easily remedied with a few lines of code.

There is so much arrogance on each side of this debate that I cannot see a reasonable solution emerging any time soon. A few researchers routinely disparage developers as incompetents whose software shouldn't be made available—even comparing developers to dogs (yes, that has happened at major security conferences). Some on the other side routinely disparage legitimate and highly valuable research as the product of a few "yahoos" who lack a moral compass or are even "criminals." There seems no likely meeting of minds between these poles.

I've set out the context in which researchers and designers exist in order to frame the important dialog between security research aimed at discovering new attack techniques and the designers of the systems and protocols upon which that research is carried out. As Adam noted so wryly, achieving solid designs, even great ones, and most importantly, resilient designs in the face of omnipresent attack requires a dialog, an interchange of constructive critique. That is how Spectre and Meltdown were discovered and presented.

Neither of this collection of (at the time of announcement) new techniques involved exercising a flaw—that is, a design error; in other words, the headlines quoted just above were erroneous and rather misleading [although salacious headlines apparently increase readership and thus advertising revenue. Hence, the misleading but emotion-plucking headlines].

Speculative execution and the use of kernel mapped user memory pages by operating systems were intentional design choices that had been working as designed for more than 10 years. Taken together, at least some of the increases in CPU performance over that period can directly be tied to these design choices.

Furthermore, and quite importantly to this discussion, these design choices were made within the context of a rather different threat landscape. That is, consider my matrices above.

Some of the actors didn't really exist, or at least, were not nearly as active and certainly not as technically sophisticated circa 2005 as they are at the time of this writing (2019).

If I recall correctly (and I should be able to remember, since I was the technical lead for Cisco's web infrastructure and application security team at that time), in 2005, network attacks were being eclipsed by application-focused attack methods, especially web attack methods.

Today, web attacks are very "ho, hum," very run of the ordinary, garden variety. But in 2005, when the first speculative execution pipelines were being put into CPUs, web applications were targets of choice at the cutting edge of digital security. Endpoint worms and gaining entrance through poor network ingress controls had been security's focus up until the web application attack boom (if I may title it so?). The web application was fast displacing these concerns as attackers shifted to targets that were always available via the Internet.

Indeed, as we learned at the time, attacks hidden within web application messages might get shuttled through applications to richer internal and backend systems, as web front ends were being attached to existing backend resources. It took a few successful compromises to understand that a web application that passes data through defenses to other systems has to act as the "firewall"—the message protection layer for those secondary and tertiary systems laying behind web layers, perhaps deep within the supposedly protected layers of an organization's network.

In other words, the threat landscape changed dramatically over the years since the initial design of speculative execution CPUs, as it must continue to evolve. Alongside the changes in types of attackers as well as their targets, attacker and researcher sophistication has grown, as has the available toolset for examining digital assets—that is, systems, software, hardware; 2018 is a different security world than 2005. I see no end to this curve of technical growth in my crystal ball.

The problem is, when threat modeling in 2005, one looked at the attacks of the past, those of moment, and tried to project from this knowledge to those of the foreseeable future. Ten or 12 years seems an awfully long horizon of prescience, especially when considering the rate at which technical change continues to take place.

Still, as new research begins to chew at the edges of design, I believe that the wise and diligent practitioner revisits existing threat models in light of developments. If I were to fault the CPU and operating system makers whose products are subject to Spectre or Meltdown, it would be for a failure to anticipate where research might lead as research has unfolded. CPU threat modelers could have taken into account advances in research indicating unexpected uses of cache memory that contains remnants of a speculative execution branch. Such examination of the unfolding train of research might very well have led those responsible for updating CPU threat models to a potential for something like Spectre and Meltdown.

Was there such research? Indeed, there was, with publications starting three years previous pointing perhaps somewhat indirectely toward the new techniques. Spectre and Meltdown are not standalone discoveries but stand on a body of CPU research that had been ongoing and published regularly for several years.

As I wrote for McAfee's Security Matters blog in January of 2018, "Meltdown and Spectre are new techniques that build upon previous work, such as 'KASLR' and other papers that discuss practical side-channel attacks. The current disclosures build upon such side-channels attacks through the innovative use of speculative execution . . . An earlier example of side-channel based upon memory caches was posted to Github in 2016 by one of the Spectre/

Meltdown researchers, Daniel Gruss."[*] Reading these earlier papers, it appears to me that some of the parent techniques that would be used for the Spectre and Meltdown breakthroughs could have been read (should have been read?) by CPU security architects in order to re-evaluate the CPU's threat model. That previously published research was most certainly available.

Of course, hindsight is always 20/20; I had the Spectre and Meltdown papers in hand as I reviewed previous research. Going the other way might be more difficult.

Spectre and Meltdown did not just spring miraculously from the head of Zeus, as it were. They are the results of a fairly long and concerted effort to discover problems with, and thus, hopefully, improve, the designs of modern processors. Indeed, the researchers engaged in responsible disclosure, not wishing to publish until fixes could be made available.

To complete our story, the driver that tipped the researchers to an early, zero-day disclosure (that is, disclosure without available mitigations or repairs) were the numerous speculative (if you'll forgive the pun) journalism (see headlines quoted above) that gained traction based upon misleading, at best, or simply wrong conclusions. Claiming a major design "flaw" in millions of processors is certainly a reader-catching headline. But, unfortunately, these claims were vastly off the mark, because no flaw existed in the CPU or operating system designs.

While it may be more "interesting" to imagine a multi-year conspiracy to cover up known design issues by evil CPU makers, no such cover up and conspiracy appears to have taken place.

Rather, in the spirit of responsible disclosure, the researchers were waiting for mitigations to be made available to customers; CPU manufacturers and operating system coders were heads down at work figuring out what appropriate mitigations might be, and just how to implement these with the least amount of disruption (see inset). None of these parties was publicly discussing just why changes were being made, especially to the open source Linux kernel.

Which is precisely what one would expect: embargo the technical details to foil attackers and to protect users. There is actually nothing unusual about such a process unfolding; it's all very normal and typical, and unfortunately for news media, quite banal. [Disclosure: I've been involved in numerous embargoed issues over the years.]

As a part of my role at the time, I was privy to the embargoed details of Spectre and Meltdown before the researchers' disclosure. Hence, I was aware of how engaged at least one CPU manufacturer was in developing fixes for the issues.

What we see through the foregoing example about Spectre and Meltdown is precisely the sort of rich dialog that should occur between designers and critics (researchers, in this case).

Designs are built against the backdrop and within the context of their security "moment." Our designs cannot improve without collective critique among the designers; that such dialog remains internal to an organization, or at least a development team, is essential. I have spoken about this process repeatedly at conferences: "It takes a village to threat model."

But, there's another level, if you will, that can be achieved for greater constructive critique. Once a design is made available to independent critics—that is, security researchers—research discoveries can and, I believe, should become part of an ongoing re-evaluation of the threat model—that is, the security of the design. In this way, we can, as an industry, reach for the constructive critique called for by Adam Shostack.

[*] Gruss, Maurice, Fogh et al. 2016.

1.4 Warlords and Pirates

"A former FBI official says the sprawling Russian black-market forum for illegal hacking and fraud services known as Infraud Organization—its motto was "In Fraud We Trust"—was operated like a 'dark-web cousin of major commercial marketplace sites.' The official said it shows one thing: that we're clearly not just fighting solo hackers at this point."*

Cybercrime, as I have been saying for some time, is a business. Although there are single practitioners, lone wolves, much of the activity is a part of larger crime organizations' business model.

No matter how comfortable any organization is with its current security posture, everyone should remember that sophisticated adversaries will be studying current practices for weaknesses, continually poking at these on a regular basis. In other words, attackers are intelligent and adaptive.

The wise security practitioner will also keep abreast of the development of analysis tools. In my first response to a security incident (I think it was circa 1992 or 1993), there were virtually no tools that might be applied beyond those used to develop software, source and assembly debuggers, binary file editors, etc. I ended up removing a worm by manually rewriting a Macintosh executable's process jump table (the offsets in the binary file at which various functions happened to be stored after the linking process). Luckily, by that time, I possessed enough computer science understanding to figure out how the worm was propagating and how to stop the propagation.

Today's toolset makes my 1993 collection look like paleolithic stone tools by comparison. One can stop a binary for which one has no source code upon any logic condition. Tools will attempt to decompile the code back to a reasonable approximation of the original source code. One can inject code into the binary and run scripts based upon data conditions, poke and prod the code and data nearly as one wishes. It's a completely different ballgame, affording attackers and researchers far more access, far more information than we dared to dream might be possible in the foreseeable future back in 1993.

We felt lucky enough when we could get source debugging to help us figure out coding problems. Today, that's almost a given, even if the decompilation is merely an approximation. The available tools are rich in functionality and deep in analysis in the hands of a skilled technician. To paraphrase myself, "Any software that can be engineered by humans can be reversed by humans." Those of us on the defensive side must understand the level of adversary sophistication brought against our defenses.

Given a rich set of available resources, it should be no surprise that some humans would wish to take advantage of others. Humans have been taking advantage of other humans ever since the first band of humans figured out that stealing from their neighbors, and quite possibly, getting those neighbors to do much of the dirty work of life (we call it "slavery" today) was easier than eking out a living through some labor-intensive combination of hunting, gathering, horticulture, animal stewardship—whatever the local mix might be. I'll opine that thieving is as much a human activity as is toil; humans seem to me to be quite good at inventing a rationale as to why theft has moral validity: "We're civilized, they aren't," "We're civilizing them," "We're smarter," "We're 'humans', they aren't"—pick your favorite justification; they've all been tried. Cybercrime seems to be yet another in the long series of opportunistic, if highly amoral (to my sense), manner of getting.

* Vaas, 2018.

To me, cybercrime seems precisely analogous to piracy. Or rather, cybercrime may very well be the 21st century's version of piracy. Cybercrime attacks have the same opportunistic, hit-and-run quality. The Internet is a common resource, big enough to offer a similar type of anonymity as was once provided by the oceans. There are safe-haven localities that are loath to prosecute cyber criminal activity, just as there once existed pirate cities, ports, enclaves, and islands where stolen goods could be traded and some much-needed rest and relaxation could be had for sea-tired crews.

Analogously, pirates also formed teams, partnerships, even navies under the command of a single leader. We have exactly that scenario today: major criminal networks retain cybercrime divisions. These criminal enterprises can garner significant revenue; it's big (criminal) business.[*][†]

But cybercrime activity is not limited solely to criminal networks. Or rather, the distinction between crime for purely business gain and that for national interest is fuzzy at best for some countries and some gangs.

"The North Korean government uses a shadowy network of cyberactors to conduct financial crimes on behalf of Kim Jong Un's regime that have attempted to steal over $1.1 billion in 'particularly aggressive' attacks on global banks."[‡]

North Korea may be at the far side of a continuum between purely state-sponsored crime and purely business driven. Still, other countries have made use of purely criminal enterprises, quasi-governmental groups, and so forth; the picture is indeed fuzzy (see inset). If the attacking organization is large enough, I think of these as governed by a "warlord"—that is, by a person or persons who maintain a private, non-governmental army which is used for the enrichment of the warlord and her/his/their retainers.

The current picture is again analogous to the famous pirates of the Barbary Coast during the 17th–19th centuries. Governments along the North African Mediterranean coast offered shelter to pirates or even allowed their own navies to pirate so long as the tax on pirated booty was properly paid.

The upshot for those of us not aligned with a warlord or pirate navy, not conducting governmental or quasi-governmental cyber operations, is that we're the collateral damage of a very confusing mix of governments and gangs sometimes operating independently and sometimes coordinating. Ugh. It's not pretty.

Most importantly, these actors have at their disposal:

- Weaponized attack code
- Exploit kits (EK)
- A rich and robust vulnerability and exploit development tool set
- A burgeoning knowledgebase on cyber attack techniques
- A flourishing marketplace in exploits and cyber attack services

Which all comes together to make a defender's life "interesting," at best. Welcome to my world, and the world of what Gary Berman (entrepreneur and comic book author) calls, *The Cyber Heroes,* those who've dedicated themselves to protecting "us" from becoming further collateral damage in an ongoing cyber war (even if not all involved are state actors).

[*] Ismail, 2015.
[†] FBI, 2019.
[‡] Cohen, Marquardt, and Crawford, 2018.

1.5 What Is the Scope of a Security Architect?

Security architecture—that is, the application of information security to systems—may be called *the art of securing systems.* As we will see in Chapter 2,[*] security architecture applies a particular set of knowledge to systems—relevant attacks and the defenses that will mitigate or, hopefully, prevent those attacks that seem relevant from succeeding.

1.5.1 Are There Really Two Distinct Roles?

In *Securing Systems,* I made a distinction between the practice of security architecture as applied to systems that aren't to be a part of an organization's protections and to those that are intended to form a defense. In practice, this distinction exists, and practitioners sometimes, perhaps often, specialize in one side of the art or the other. That is, security architects might be specialists in building an organization's defenses and reactive structures, the organization's security *architecture.* Alternatively, an architect might specialize in analyzing systems to identify security needs, security *requirements* that should be built in order to protect the system and its organization's goals for that system.

For instance, The Open Group proposed a number of security reference architectures (see inset) that directly address an organization's need for sound advice on how to build a set of defenses that rest on strong and battle-tested security principles.

As far as I know, The Open Group's only offering in the system analysis arena is Factor Analysis of Information Risk (FAIR): a risk standard that can be applied to systems, and really, any digital security problem.

The last time I checked the progress of The Open Group's reference architectures for security, these had been integrated into a set of enterprise reference architectures. I believe that this is an important step, reflecting how the enterprise security architect should approach the problem of a security architecture: as a key component of the enterprise's architecture—"enterprise" in this use should be taken as equal to "organization of sufficient size to warrant an organization architecture," which is a much broader categorization than the commonly used definition of "enterprise."

Risk Must Be Fundamental

Surely, every system analysis for security, threat model, should be based upon a solid risk methodology such as FAIR, which is my personal favorite as well as the theoretical basis for Just Good Enough Risk Rating (JGERR),[†] authored by myself and Vinay Bansal (Distinguished Engineer at Cisco Systems, Inc.). JGERR may be thought of as a quick-and-dirty child of FAIR meant for the dozens of quick risk ratings that usually come up during threat modeling.

Whatever approach a security architect chooses to base risk ratings upon, it must have a firm theoretic basis. Too often, risk is measured by the discomfort of the analyst, which leads to mushy, inconsistent

[*] Chapter 3 will provide an analysis of attacks and defenses for a well-known vulnerability, Heartbleed to put the technical flesh on the bones of what is presented in Chapter 2.

[†] JGERR is described in some depth in *Securing Systems,* Chapter 4.

risk ratings, which then may propagate into poor organization risk metrics, usually inflating the metrics, leading to a sense of an increased, perhaps unreal amount of risk being carried.

Teams subject to poor risk rating methods may "shop" for the best rating, because they don't trust the higher, perhaps inflated ratings. Or, as Jack Jones (author of FAIR) once told me, executives may feel inclined to accept nearly every risk in the absence of solid and consistent risk assessment.

Readers can probably draw the line from poor risk rating to organizational distrust of security architecture, maybe even all of the security function? Certainly, organizational friction lies down that slippery slope. As you may see, choice of risk rating methodology is a critical component for any mature and robust security practice; The Open Group is to be applauded for standardizing FAIR and making it available to organizations.

Still, risk assessment is only one portion of the security architecture of systems large and small, of system assessment for security. Since The Open Group hasn't had much help in this area, the vacuum has been filled by materials from the Open Web Application Security Project (OWASP),[*] SAFECode,[†] and similar organizations. Of course, a few practitioners have tried to set down their thoughts, methods, and experiences for threat modeling in a few books, including yours truly (please see bibliography).

I've personally lived both of these roles in my career; the distinction exists, but I wonder if this distinction is more an artifact of organization structures rather than a real divergence in practice.

Analyzing a discreet system or set of systems performing a particular function indeed requires a different focus from thinking through the structure of an organization's entire security implementation. Still, in both instances, one must consider the sorts of attacks most likely to occur and how these will be prevented, or if successful, dealt with. The art which ties both these strands together is the art which we have proposed here to lie at the heart of security architecture: attacks and their defenses.

It seems to me that the different specialties in security architecture appear to be more a difference in kind, in degree, in scope, rather than some fundamental split in practice. Both analyses must base themselves firmly upon solid risk methodology. Both must understand the desired risk posture of the organization as well has have a reasonable feel for the amount of risk the organization is willing to carry ("risk tolerance").

Again: Attacks and Defenses

For system assessment, one must consider attacks relevant to that system *in the context of the security architecture,* if any, that may surround that system. To build a security architecture for an organization, the security architect must consider the attacks to which *any and all* systems of the organization may be subject. This organization analysis should be more holistic to the threat landscape in which the organization exists.

On the other hand, every system analysis must be holistic to that system. The details of existing network protections, incident response capabilities, access controls, etc. will be taken into account in both analyses, what I called "Mitigations" in the "ATASM" mnemonic (Architecture,

[*] OWASP.org

[†] The author is a co-author of SAFECode's Threat Modeling Guide

Threats, Attack Surface, Mitigations): the existing protections. Such protections would be a part of the organization's security architecture. I hope that you see that these two seemingly distinct practices are really views of the same coin from its different faces.

The analysis that leads one to the contextually relevant attacks, let's call that "threat modeling" for the sake of discussion, is best done as an early part of any development process, as well as being an ongoing conversation as architectures, designs, and implementations evolve. Certainly, there is very little that an architectural risk assessment of a system can do if the system cannot be changed (see inset). Consequently, threat modeling is an activity that starts early in the development cycle.

I do not mean to imply that the risk assessment portion of a threat model of a production system that has no further intended changes is useless. Risk assessment can help an organization build a picture of risk "debt"—the risks that have already been taken and are carried forward. At the very least, building this risk knowledge may help make better decisions about what further risk to add to existing risks carried forward.

Patterns, Standards, and Context

The art of architecture involves the skill of recognizing and then applying abstract patterns while, at the same time, understanding any local details that will be ignored through the strict and inflexible application of patterns. Any unique local circumstances are also important and will have to be attended to properly. It is not that locally specific details should be completely ignored; rather, in the interest of achieving an "architectural" view, these implementation details are overlooked until a broader view can be established. That broader view is the architecture.

There is a dance between adhering to standards and fostering innovation. New technologies come along that disrupt standards. These innovations may provide significant benefit if adopted. Usually, there are early adopters who help prove the usefulness and benefits (or not) of new technologies. The successes of the early adopters help to drive adoption through an organization.

"Computer security exists as an attribute, an emerging property (or not!) of systems that exist within an extremely rapidly changing context, that is, digital technology. It is simply too difficult to anticipate all circumstances, external and internal, in a policy at this time. Rather, policy becomes the bedrock to which many systems can mostly conform. The standards that set out how the policy will be enacted create an easy path, a reasonably secure path that can be followed. At the same time, policy and standards help to define areas and situations that will require creativity—those systems, connections, technologies that are necessary, but which cannot conform: the exceptions to standards. Sometimes, these exceptions will reduce the security posture. Sometimes, exceptions will offer an opportunity to mature security in new ways or open opportunities to adopt new technologies." *Securing Systems,* p. 354

Imagine an organization that insists upon strict adherence to a standard of one application server per application. That was a very durable model in the mid-2000s. That model forced an operating system and application server sandbox around each application (or set of interoperating "applications"). In the days before proven web application firewalls and in the context of poor understanding by developers of secure web coding techniques, strictly separating applications at a level below the application code made a lot of security sense. At the very least, a compromised application couldn't disrupt other applications; the sandbox prevented the attacker from breaching each application's sandbox boundaries.

One of the downsides engendered by an application server sandbox was that providing each single application with its own, individual application server required a rather large investment in virtual machines on top of significant physical server resources. That was *the* cutting-edge model at the time. There was no cloud into which to expand; clouds as we now know them didn't exist.[*]

If an organization insisted upon never allowing any new technologies beyond the model described above, that organization would have missed containerization completely. DevOps tooling tends to be built around containers and/or clouds (which also can offer highly containerized solutions). Consider just those two developments: They've greatly disrupted the architectures and methods by which we deploy web applications. The movement to serverless architectures is a currently (as of this writing) unfolding disruption.

Failure to account for experimentation with new and disruptive technologies is a massive error in the service of a surer security path. It is my strong experience and opinion that, alongside standards that make the "easy path the secure path,"[†] the mature—actually, the wise—organization provides for experimentation with new, potentially disruptive technologies and techniques.

However, the wise security architect will bear in mind that sometimes a developer does not follow the easy path to security but employs a new technology in a place where it is not needed or does not really fit. The developer insists upon the use of the inappropriate technology so that it may be included on her, his, their resume. This use case has nothing to do with the fitness of the technology or the "easy" path; the use is intended to enhance the programmer's career.

Identifying an inappropriate application of a technology can be tricky. One of the obvious giveaways is if the requested technology will have obvious negative effects. For instance, many years ago, when web services using SOAP (Simple Object Access Protocol [XML protocol]) were in vogue, I found a few teams using them when doing so might add significant performance degradation. SOAP calls must all occur in ASCII (American Standard Code for Information Interchange). Binary data must be converted to ASCII for transmission over SOAP and then converted back to binary after communication. That may make perfect sense for transactions that occur at human pace, but it's a huge performance hit for data exchanges meant to proceed as quickly as computers can process them.

ASCII conversion can cause data to balloon up to eight times larger. So, such a conversion should be thought through thoroughly. It's particularly troubling when both sides of the web service, client and server, are to be located on the same machine, when the SOAP server is implementing a simple, atomic Application Programming Interface (API). SOAP services should be transactional, not atomic, if possible.

These were all red flags to me that somehow the proposed design didn't make sense. After all, simply linking a library, either statically as a part of the executable or dynamically, would be orders of magnitude faster. Data could be exchanged with a library in its binary form without

[*] Or rather, clouds as we use them today were in consideration, a few were being designed.

[†] As Steve Acheson so sagely advised me many years ago.

any conversions. When reviewing a design that screams, "there is a simpler, more effective way that is well understood," the security architect may well have encountered one of these situations in which there is another reason (often that cannot be named safely) for making poor choices.

When my "Spidey sense" is screaming that something doesn't make sense, I've now learned to step back from security issues to question what the reason might be for doing something that obviously doesn't make design sense. Quite often, the designer has ulterior motives, like trying to squeeze in technologies that will bolster her or his resume.

What does one do in such a situation? I question directly the choice and my reasons why it appears to be odd or just plain wrong, why there appears to be an easier or more elegant solution. Sometimes, that's effective. But, in some organization cultures (highly empowered at the team level) there may be nothing one can do about it.

If reasoning with the team fails, one can always go up the team's management chain. Also, the executives who are sponsoring the development, or who are the customer, may wish to understand that I have serious concerns about the way the software is designed. I've won internal awards for exposing poor design choices (see inset).

It's one thing to blow the whistle on a situation that one believes might expose the organization to poor performance or higher costs. It's quite another when dealing with those who are solely working for personal benefit. That is, when one is working with those of low moral standard, who lack integrity, proceed with caution! Exposing people of this nature to management can be tricky; one may expect such individuals to protect themselves with every tool they can muster. An unmask must be thought through; who protects the person? Who are my allies and what is their relation, if any, to the project in question? I do try to prevent great harm from taking place. But I also try to remember that low-integrity people may move to something new soon enough to give me a greater hand to undo damage, or they may fail of their own accord. This situation hasn't arisen often in my 30+ years in high tech, but it has come up a few times, which has been a great teacher about organization politics.

As always, analysis and discernment are critical: I don't want to stop useful experimentation with new, promising technologies. At the same time, if something doesn't seem right, I believe it's my duty to question choices, even if I must step beyond the scope of security.

At the most essential, all security architecture activity can be reduced to analysis of attacks, rated by solid risk rating, leading to defenses, as we shall hopefully see in the next section and throughout this book in various ways.

1.6 Essential Technique

1.6.1 Threat Modeling: An Essential Craft

Attacks and their defenses are the value proposition that security architects bring to the design and implementation of software—that is, development, engineering, or research and development. Over the years, "threat modeling" has gone by various names:

- Architecture risk assessment (ARA)
- Architecture review or security architecture review

- Security architecture assessment
- Security engineering
- [Secure] design review
- Secure design checkpoint
- Security requirements

And quite possibly, a few other terms that are, by now, lost in the mists of time.

Although practitioners are certainly free to disagree with my collapsing all the above terms into one analysis that in their processes seem quite distinct, I came to realize how threat modeling analysis actually rather completely underlies what appear to be different analyses. Threat modeling is the method that practitioners must apply no matter at what point in development a security analysis is taking place, no matter whether completed or initial or inflight a project or effort may be.

For a couple of years, my duties as a Principal Engineer for software security at Intel® included sitting on Intel's software security review panel, SAFE (Security Architecture Forum). We reviewed projects from initial concept through the completion of the design. At Intel, there are several review stops for security; more or less the same body of people would engage across projects and at these different review points during development.

Of course, I was a full participant, attempting to apply my best understanding of security architecture to each project as it came before the SAFE board. At the same time, as I often do, I was a participant observer, considering what I heard during the interactions between development teams and board members, assessing the efficacy of our process (or not) as reviews proceeded. I found the experience of watching my peers—that is, other Principle Engineers—practice security architecture very enlightening—not only to refine my own craft, but also in stepping back from the content of the work to observe how we do what we do. The projects ranged across myriad and often vastly different architecture types, projects at every stage of development, using nearly every type of software development methodology (waterfall, Agile, Extreme, etc.) (see inset). SAFE would typically interact with two to four projects each week of the year. That's a lot of projects in any given month, quarter, or year to observe and from which to learn.

I encourage every serious practitioner to work at a really large development organization at least once in a career. First, one gets to rub shoulders with some of the very best in each discipline. Further, one learns how to meet the challenges of scale: huge projects combining the work of many sub-teams, application of security methods to vastly different development approaches, and, perhaps most importantly, finding the essences that lie beneath or at the heart of varied expressions of technique and process.

As I watched the SAFE review process unfold, it became crystal clear to me that no matter at what stage or in which review point we board members were with a project, in order to complete the review, we were all threat modeling in our heads, call the review what you will. This was quite a revelation.

What differed was the level at which the threat model analysis occurred.

For instance, during an official threat model review, we had to dig deep, to attempt to cover every credible attack via every reachable attack surface (exposure) and find reasonable, workable, implementable defenses that could be built by the team presenting their project.

However, if a project had just been initiated, the level of threat modeling was vastly different. All we needed was to think through a few gross possibilities in order to derive the very broad security requirements and to understand the risk posture that the effort would need in its usage and deployment context.

The architecture assessment phase in the Intel Secure Development Lifecycle (SDL) is intended to identify those efforts that will require deeper security analysis, while at the same time passing those efforts whose security architecture needs will likely be (comparatively) minor. The threat model analysis for this review needs only to determine the credibility of a significantly impactful successful attack. One credible attack that might cause Intel's or the product's stakeholders significant harm is all that was required to flag the project for further engagement. The analysis technique is threat modeling, nevertheless.

I don't mean overstate the importance of threat modeling, call it what you will. Rather, as I've written and as I hope that you see in succeeding chapters in this book, security architects must wield attacks and their defenses as a primary knowledge set that is applied to software systems. Threat modeling lies at the heart of the practice of security architecture.

But threat modeling is not the only skill that is applied. Security architects, as I have written, must first and foremost be architects.

1.6.2 Architecture Is Primary

"I would suggest that architecture is the total set of descriptive representations relevant for describing something, anything complex you want to create, which serves as the baseline for change if you ever want to change the thing you have created."[*]

I would go further than Zachman to state that understanding, ability to discuss, and potential change are the benefits derived from architecture.

To understand a thing, we must be able to describe it in simple enough terms to grasp it; to discuss it intelligently, we must be able to name its parts, explain their function and their interactions. That is what architecture affords us through the clever and judicious application of abstraction. I like to explain that architecture is a practice whose main tool is abstraction.

In order to reduce complexity sufficiently for comprehension, architects abstract structures for analysis while obscuring detail which may mask underlying structure or which is not relevant to structural comprehension. This is abstraction: drawing out some information in favor of other information that is unnecessary. Identifying and (conceptually) manipulating structures is the goal; abstraction is the technique of architecture.

Hence, security architects must be skilled in highlighting those structures that have security implications (typically, functions of a system, its software units of implementation, often called "components," and the communications between these structural elements). Like any architect, we may obscure or elide (leave out) extraneous detail that is unnecessary to understanding and analysis (the analysis is often threat modeling). By the application of judicious abstraction, security architects are no different than any other type or level of system, software, integration, or enterprise architect.

Because the nature of the analysis (attacks and defenses) is different, what structure is abstracted and what detail eliminated may differ, even radically, from that which is useful to

[*] Zachman, 2007.

architects charged with other aspects of a system's developing architecture. But the mental process is the same, even if the working diagram differs.

1.7 Aiming Design Toward Security

Although any particular branch of architectural analysis may be focused on different results, there is one aspect of the practice of architecture that must be precisely the same: We all have to have a firm and precisely communicable idea about what must be achieved by a system. In order to architect, we have to know what we are building, what goals we are trying to achieve.

1.7.1 What Is Secure Software?

In the practice of security architecture, we must then understand what software security looks like. What are the behaviors that secure systems must exhibit? How is a "secure system" defined?

Over the years that I've been practicing, as I open a discussion about security with development teams, I've noticed that quite often (not every time, but regularly), team members will immediately jump to one of four aspects of software security:

- Protection of data (most often via encryption techniques)
- Implementations errors (most often, coding securely)
- Authentication and/or authorization of users of the system
- Network-level protection mechanisms

This set of responses has been remarkably stable for the last nearly 20 years, which is interesting to ponder all by itself. Despite the dramatic shift in attacker capabilities and techniques over the last 20 years—a huge shift in attacker objectives—developers seem to be thinking about one of the above aspects of the security picture. I don't know why development has not kept pace with the expansion of adversarial thinking, but apparently it hasn't (though, of course, my evidence here is completely anecdotal and not at all scientifically validated).

Lately in my threat modeling classes (and sometimes other presentations), I've been polling my audiences about what jumps first to mind when I say, "software security." Not surprisingly, members of my audiences typically find themselves considering one of the above categories unless a participant has broader security exposure. My informal polls underline a need to establish a baseline definition of just what software security must include, the field's breadth, its scope.

To address the challenge that development teams often lack a sufficiently complete picture of what software security entails, as well as to provide a set of secure design goals, I came up with the following secure software principles. "Secure" software must:

- Be free from implementation errors that can be maliciously manipulated: ergo, vulnerabilities
- Have the security features that stakeholders require for intended use cases
- Be self-protective; resist the types of attacks that will likely be attempted against the software
- In the event of a failure, must "fail well"—that is, fail in such a manner as to minimize consequences of successful attack
- Install with sensible, "closed" defaults

The foregoing are the attributes that “secure software” displays, to one extent or another, as it runs. These principles are aspirational, in that no running system will exhibit these behaviors perfectly; these cannot be implemented to perfection. Indeed, so far as exploitable conditions are concerned, whether from implementation, from a failure to identify the correct security requirements, or a failure to design what will be implanted correctly, software, at its current state of the art, will contain errors—bugs, if you will. Some of those errors are likely to have unintended security consequences—that is, vulnerabilities allowing adversaries leverage or access of one kind or another. This truism is simply a fact of building software, like it or not.

Then there is the matter of security context and desired security defensive state: a system or organization’s security posture. Not every system is expected to resist every attack, every adversary, every level of adversary sophistication and level of effort that can be expended (given the universe of various threat agents; please see my discussion of adversary types elsewhere in this book as well as in *Securing Systems*).

Hence, presence and robustness of the above secure software behaviors must vary, system to system, implementation to implementation.

Still, I expect software to account for the above behaviors, even if by consciously accepting the risks generated by a considered absence or weakness of one or more of the principles given above. My software principles are meant to drive secure design decisions, to be goals to reach for. None of these principles is built as stated. These principles don’t tell you how to protect a credential that must be held by a system. Rather, from these principles, design choices can be evaluated. These are guideposts, not design standards.

1.7.2 Secure Design Primer

We know with fair certainty that credentials (secrets) that are placed in the binary executable of software are relatively easy to uncover, given today’s reverse engineering tools. For commercial software whose distribution involves execution within third-party environments—say, commercial, off-the-shelf (COTS) software that is run by the purchaser, distributing a secret tucked away in the static data of an executable—has proven to be a very poor design choice.

Such packaged secrets are routinely discovered by both attackers and researchers. Once held by an attacker (whether through discovery or published research), the attacker then has the ability to wield the secret successfully against whatever challenge it was meant to protect. If there are 10,000 copies of that executable in use, attackers can undo at least some aspect of the security for all 10,000 of those installations. Unfortunately, this design mistake happens far too often, with the resulting consequences.

How do our secure software principles apply? If programmers have coded the credential into the binary with correct language semantics (easily coded—for most languages, this is just a declaration), then there is no “implementation error.” This is a design error. It’s a failure to have the security features that stakeholders expect—that is, credentials have sufficiently been protected.

Furthermore, attempting to “hide” a secret as static data in an executable isn’t self-protective. Quite the reverse, given the binary exploration and execution analysis tools that exist today.

Often this design miss assumes that the credential is “safe enough,” so the use of the credential (which is legal) and the actions taken after the challenge has been passed will not be monitored; there is no failure—the credential is working as expected, but in the hands of adversaries.

So the "fail well/fail closed" principle doesn't really apply. It would depend upon the installation and configuration sequences of the software as to whether the placement of the credential in the binary is a "sensible, closed default."

I hope that the preceding trivial example demonstrates how the software security principles are meant to provide appropriate targets for deriving a secure architecture. The design patterns that will achieve these results require quite a bit more detail, which then must be applied in context.

A high-level set of secure design patterns and their application can be found in IEEE Center For Secure Design's "Avoiding the Top 10 Software Security Design Flaws."[*] This booklet is free, under The Creative Commons license. (Disclosure: I'm one of the the co-authors.[†]) The design patterns discussed in the booklet are:

- Earn or give, but never assume, trust
- Use an authentication mechanism that cannot be bypassed or tampered with
- Authorize after you authenticate
- Strictly separate data and control instructions, and never process control instructions received from untrusted sources
- Define an approach that ensures all data are explicitly validated
- Use cryptography correctly
- Identify sensitive data and how they should be handled
- Always consider the users
- Understand how integrating external components changes your attack surface
- Be flexible when considering future changes to objects and actors

Each of the design patterns explained in the booklet is meant to fulfill one or more of the software security principles listed above. First, we define how we intend secure software to behave (secure software principles) and then set out the means through which those aims are to be achieved (secure design patterns). With software security principles and design patterns, we know what we are to build when we wish to build "secure software."

1.8 Summary

We've explored the societal context of an unfolding "Computer Age" and the effects on the people in what I call the "connected life." This context points to a need for a general improvement in computer security if we are not to fall prey to cyber pirates, warlords, and armies. In order to build sufficient security into software, there exists a specific job role: the security architect. This book attempts to address what security architecture practice is, what we do (essentially, threat modeling) and how we do it, and to offer a few tricks of the trade achieved through many mistakes and missteps, a great deal of help from brilliant practitioners, and a lot of diligent practice.

[*] IEEE, 2014.

[†] Iván Arce, Kathleen Clark-Fisher, Neil Daswani, Jim DelGrosso, Danny Dhillon, Christoph Kern, Tadayoshi Kohno, Carl Landwehr, Gary McGraw, Brook Schoenfield, Margo Seltzer, Diomidis Spinellis, Izar Tarandach, and Jacob West.

The following chapters examine more closely the art and some considerable computer science of attacks and defenses. As well, I will try to offer sufficient views of security architecture such that it may finally be more firmly defined. Once in command of what security architecture might actually be, the remainder of the book will concern itself with the practice thereof, from the perspective of the learner, the practitioner, and the strategist. I hope that these explanations will prove useful to your practice.

References

Alperovitch, D. (2011, August 2). Revealed: Operation Shady RAT. McAfee, Inc. White Paper.

Greenberg, A. (2018, January 3). A Critical Intel Flaw Breaks Basic Security for Most Computers. *Wired Magazine.* Retrieved from https://www.wired.com/story/critical-intel-flaw-breaks-basic-security-for-most-computers/

Bonardon, O. (2018) Retrieved from https://docbox.etsi.org/Workshop/2018/201806_etsisecurityweek/middlebox/s03_joint_efforts/encrypted_traffic_inspection_mcafee_bonorden.pdf

Bureau of Labor Statistics. (n.d.). US Dept. of Labor Statistics for Job Class "Security Analyst." Retrieved from https://www.bls.gov/ooh/computer-and-information-technology/information-security-analysts.htm

Cerbin, W. (2011). Understanding Learning Styles: A Conversation with Dr. Bill Cerbin. Interview with Nancy Chick. UW Colleges Virtual Teaching and Learning Center.

Chandra, B. (2014, May 13). A Technical View of the OpenSSL "Heartbleed" vulnerability, Version 1.2.1. A Security on DeveloperWorks Community Whitepaper. ibm.biz/dwsecurity. Retrieved from https://www.ibm.com/developerworks/community/files/basic/anonymous/api/library/38218957-7195-4fe9-812a-10b7869e4a87/document/ab12b05b-9f07-4146-8514-18e22bd5408c/media

Cohen, Z., Marquardt, A., and Crawford, J. (2018, October 3, 12:44 PM ET). North Korean Hackers Tried to Steal over $1 Billion, Report Says. CNN. Retrieved from https://www.cnn.com/2018/10/03/politics/north-korea-hackers-cybercrimes/index.html

Doyle, J. (1998). *Routing TCP/IP,* Vol. I, Cisco Press, MacMillan Technical Publishing, p. 41.

Ducklin, P. (2014, April 8). Anatomy of a Data Leakage Bug—The OpenSSL "Heartbleed" Buffer Overflow. Naked Security by Sophos. Retrieved from https://nakedsecurity.sophos.com/2014/04/08/anatomy-of-a-data-leak-bug-openssl-heartbleed/

Eadicicco, L. (2014, May 15). Photos of The NSA's Secret Workshop Where It Intercepts Packages and Plants Bugs in Electronics. *Business Insider.* Retrieved from https://www.businessinsider.com/nsa-tao-intercepting-packages-2014-5

Edwards, B. (1989). *Drawing on the Right Side of the Brain.* Tarcher-Perigree.

FBI. (2019, April 22). 2018 Internet Crime Report, Federal Bureau of Investigation, Cyber Division. Retrieved from: https://pdf.ic3.gov/2018_IC3Report.pdf

Fruhlinger, J. (2018). The Mirai Botnet Explained: How Teen Scammers and CCTV Cameras Almost Brought Down the Internet. Retrieved from https://www.csoonline.com/article/3258748/the-mirai-botnet-explained-how-teen-scammers-and-cctv-cameras-almost-brought-down-the-internet.html?

Fuller, M. (2008). *Software Studies: A Lexicon,* p. 170. MIT Press.

Ghaznavi-Zadeh, R. (2017). *ISACA Journal,* Vol. 4. Retrieved from https://www.isaca.org/Journal/Archives/2017/Volume-4/Pages/enterprise-security-architecture-a-top-down-approach.aspx?utm_referrer=&utm_referrer=

Grobman, M. (2016, November 3). Focus Keynote, Chief Technology Officer (CTO), McAfee, Inc. Focus Conference, Las Vegas NV.

Gruss, D., Maurice, C., Fogh, A., et al. (2016). Prefetch Side-Channel Attacks: Bypassing SMAP and Kernel ASLR. Graz University of Technology, G DATA Advanced Analytics. Retrieved from: https://gruss.cc/files/prefetch.pdf

IEEE (2014). Avoiding the Top 10 Software Security Design Flaws. IEEE Center for Secure Design, p. 2. Retrieved from https://cybersecurity.ieee.org/blog/2015/11/13/avoiding-the-top-10-security-flaws/

Intel. (2015). Protect, Detect, Correct: Security Connected for Healthcare Providers. Intel Security. Retrieved from http://www.mcafee.com/us/resources/brochures/br-protect-detect-correct-security-connected-healthcare.pdf

Isaac, M. (2016, October 25). Self-Driving Truck's First Mission: A 120-Mile Beer Run. *New York Times.* Retrieved from https://www.nytimes.com/2016/10/26/technology/self-driving-trucks-first-mission-a-beer-run.html

Ismail, N. (2018, April 24). Global Cybercrime Economy Generates over $1.5TN, According to New Study. *Tech Nation.* Retrieved from https://www.information-age.com/global-cybercrime-economy-generates-over-1-5tn-according-to-new-study-123471631/

Johnson, S. (1836). *The Poetical Works of Alexander Pope, Esq., to Which Is Prefixed: A Life of the Author,* Vol. 1, p. 89. J. Gladding & Co.

Kocher, P., Horn, J., Fogh, A., et al. (2018). Spectre Attacks: Exploiting Speculative Execution. Retrieved from https://spectreattack.com/spectre.pdf

Leyden, J. (2005, August 3, 15:37). Cisco Portal Password Security Compromised. Precautionary Reset Fails to Run Smoothly. Retrieved from https://www.theregister.co.uk/2005/08/03/cisco_password_security_flap/

Lipp, M., Schwarz, M., Gruss, G., et al. (2018). Meltdown: Reading Kernel Memory from User Space. Retrieved from https://meltdownattack.com/meltdown.pdf

Maruoka, A. (2011). *Concise Guide to Computation Theory,* p. 167. Springer-Verlag.

McClue, S. (1999, September 10). Hacking Exposed: Network Security Secrets & Solutions. *Computing.* McGraw-Hill.

McGeehan, R. (2019, April 23). Describing Vulnerability Risks. Medium.com. Retrieved from https://medium.com/@magoo/describing-vulnerability-risks-3a78c2e352d8

Mehta, N. (2014, April 8, 1:08 PM). Twitter.

Michlin, I. (n.d.) DevOps Trainer and Principal Security Consultant at NCC Group, quoted by Robert Lemos. Threat Modeling and DevOps: 3 Lessons from the Front Lines. Tech Beacon. Retrieved from https://techbeacon.com/security/threat-modeling-devops-3-lessons-front-lines

Microsoft. (n.d.) Simplified Implementation of the SDL. Microsoft SDL documentation, p. 6. Retrieved from: https://download.microsoft.com%2Fdownload%2FF%2F7%2FD%2FF7D6B14F-0149-4FE8-A00F-0B9858404D85%2FSimplified%2520Implementation%2520of%2520the%2520SDL.doc&usg=AOvVaw3UCYwxZwcaPbQoTPkFLQ1Q

Microsoft (2014, June 13). The OSI Model's Seven Layers Defined and Functions Explained, Rev. 2. Microsoft Inc. Retrieved from https://support.microsoft.com/en-us/kb/103884)

MITRE. (n.d.). CWE-888: Software Fault Pattern (SFP) Clusters. MITRE Corporation. Retrieved from https://cwe.mitre.org/data/graphs/888.html

NATO. (1969, October). NATO Science Committee. Software Engineering Techniques. Report on a Conference Sponsored by the NATO Science Committee, p. 16. Quote from Edsger Dijkstra, Rome, Italy. Retrieved from http://homepages.cs.nci.ac.uk/brian.randell/NATO/nato 1969.PDF

NIST 800-14 in 1996. Retrieved from https://csrc.nist.gov/publications/detail/sp/800-14/archive/1996-09-03

Ogundeji, O. A. (2015, August 18). Google Launches Android One Smartphone Program in Africa. *PCWorld.* Retrieved from https://www.pcworld.com/article/2972741/android/google-launches-android-one-phone-in-africa.html

Osborne, C. (2018, January 23). Artificial Synapse Creation Makes Brain-on-a-Chip Tech Closer to Reality. Retrieved from: http://www.zdnet.com/article/artificial-synapse-creation-makes-brain-on-a-chip-tech-closer-to-reality/#ftag=RSSbaffb68

Romeo, C. (n.d.) CEO of Security Journey. Quoted by Vijayan, J. 6 DevSecOps Best Practices: Automate Early and Often. Tech Beacon. Retrieved from https://techbeacon.com/security/6-devsecops-best-practices-automate-early-often

Rosen, M. (2008, October 1). 10 Key Skills Enterprise Architects Must Have to Deliver Value. Retrieved from https://www.cutter.com/article/10-key-skills-enterprise-architects-must-have-deliver-value-469471

Schoenfield, B. (2014). Applying the SDL Framework to the Real World. In Ransome, J. and Misra, A. *Core Software Security: Security at the Source,* Ch. 9, pp. 255–324. Boca Raton, FL: CRC Press.

Schoenfield, B. and Quiroga, D. (2017, November 24). Don't Substitute CVSS for Risk: Scoring System Inflates Importance of CVE-2017-3735. Securing Tomorrow McAfee Blog. Retrieved from https://securingtomorrow.mcafee.com/other-blogs/mcafee-labs/dont-substitute-cvss-for-risk-scoring-system-inflates-importance-of-cve-2017-3735/

Schoenfield, B. (2015). *Securing Systems: Applied Security Architecture and Threat Models.* Boca Raton, FL: CRC Press.

Seggleman, R., Tuexin, M., and Williams, M. (2012, February). Request for Comment RFC 6520 "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension." ISSN: 2070-1721. Retrieved from https://tools.ietf.org/html/rfc6520

Shankland, S. (2014, April 8, 2:55 AM PDT). "Heartbleed" Bug Undoes Web Encryption, Reveals Yahoo Passwords. Retrieved from https://www.cnet.com/news/heartbleed-bug-undoes-web-encryption-reveals-user-passwords/

Shankland, S. (2014, April 8, 2:55 AM PDT). Cost of a Retail Data Breach: $179 Million for Home Depot. WebTitan. Retrieved from https://www.webtitan.com/blog/cost-retail-data-breach-179-million-home-depot/

Shostack, A. (2014). Security Engineering: Computers versus Bridges. Adam Shostack and Friends. https://adam.shostack.org/blog/2018/04/security-engineering-computers-versus-bridges/

Stephenson, N. (1995) *The Diamond Age.* Bantam Books.

Swiderski, F. and Snyder, W. (2004, July 14). Threat Modeling. *Microsoft Professional.* Microsoft Press.

Tarandach, I. and Schoenfield, A. (2019, May 30). *Continuous Threat Modeling Handbook.* continuous-threat-modeling/Continuous_Threat_Modeling_Handbook.md, GitHub. Retrieved from https://github.com/Autodesk/continuous-threat-modeling/blob/master/Continuous_Threat_Modeling_Handbook.md)

The Open Group. (2011). An Example of Enterprise Security Thinking Can Be Found in Open Enterprise Security Architecture. Retrieved from https://publications.opengroup.org.G112

The Open Group. (n.d.) The Open Group TOGAF Standard, Version 9.2. Retrieved from https://pubs.opengroup.org/architecture/togaf9-doc/arch/chap03.html

Tipton, H. F. (2000, October 20). *Information Security Management Handbook,* 4th Edition, Vol. 2, p. 581. Boca Raton, FL: CRC Press.

Vaas, L. (2018, February 26). In Fraud We Trust—Cybercrime Org Bust Shows We're Fighting Pros. Naked Security by Sophos. Retrieved from https://nakedsecurity.sophos.com/2018/02/26/in-fraud-we-trust-cybercrime-org-bust-shows-were-fighting-pros/

Vanhoef, M. (2017). Key Reinstallation Attacks: Breaking WPA2 by Forcing Nonce Reuse. [Discovered by Mathy Vanhoef of imec-DistriNet, KU Leuven]. Retrieved from https://www.krackattacks.com/

Williams, C. (2018, January 2). Kernel-Memory-Leaking Intel Processor Design Flaw Forces Linux, Windows Redesign. *The Register.* Retrieved from https://www.theregister.co.uk/2018/01/02/intel_cpu_design_flaw/

Wilson, P. L. (1995, 2004). *Pirate Utopias: Moorish Corsairs & European Renegadoes.* AbeBooks.

Zachman, J. A. (2007). Foreword. In: *Handbook of Enterprise Systems Architecture in Practice.* (P. Saha, Ed.) pp. *xv–xvi.* IGI Global.

Zeigler-Hill, V., Welling, L. M., and Shackleford, T. K. (Eds.). (2015). *Evolutionary Perspectives on Social Psychology Evolutionary Psychology,* p. 231. Springer International Publishing.