Chapter 1

Introduction

"Who are *you*?" said the Caterpillar.

This was not an encouraging opening for a conversation. Alice replied, rather shyly, "I hardly know, sir, just at present—at least I know who I *was* when I got up this morning, but I think I must have been changed several times since then."

"What do you mean by that?" said the Caterpillar sternly. "Explain yourself!"

"I can't explain *myself*, I'm afraid, sir," said Alice, "because I'm not myself, you see."

"I don't see," said the Caterpillar.

"I'm afraid I can't put it more clearly," Alice replied very politely, "for I can't understand it myself to begin with; and being so many different sizes in a day is very confusing."

"It isn't," said the Caterpillar.

"Well, perhaps you haven't found it so yet," said Alice....

Through the Looking-Glass, and What Alice Found There
—LEWIS CARROLL

1.1 Changes

One of the most visible aspects of the computer industry is how rapidly things change. Four aspects of the change rate are of interest here: performance improvements (obvi-

ously, today's computers are much faster); capability improvements (we can do things today that we couldn't do even a few years ago); price; and environment (because people and companies around us do more, we can interact with them electronically). All of these affect security.

Recently, I received a check in the mail and deposited it by taking a picture of it with my phone. Think of the technical security challenges the bank had to deal with to make that possible:

- They have to have very high confidence that the right person is connecting to the account.
- This server application has to be very robust against all sorts of attacks; it can, after all, touch live bank accounts. In particular, it can add money to an account, based on user input; quite conceivably, their previous online application *deliberately* couldn't do that, as a security measure.
- However—the deposit is conditional, based in part on the image of the check being examined, by a human or by software, to verify the amount. In other words, some sensitive part of their system has to process an enemy-supplied image file.
- They have to allow the upload of large image files, with the consequent need for bandwidth, disk space, and more.
- My phone's operating system has to be secure enough that rogue phone apps can't spy on or modify the banking transactions.
- All traffic has to be encrypted.
- The phone has to be assured of connecting to the proper destination.
- There needs to be a proper audit trail for all transactions.
- Everything must work seamlessly with the "traditional" web application (itself not more than 15 years old, and probably a lot less), human tellers, and the legacy back-end systems that may have originally been written in COBOL and entered on punch cards for some giant mainframe, but now probably runs on a mainframe emulator on the CTO's tablet.
- Given all of these other changes, the entire architecture's security characteristics should be revisited.

Obviously, my bank and many other banks have made the necessary changes; the application works. The system architects figured out what had to be done; the security folks, the programmers, the network engineers, and everyone else made the necessary changes.

The interesting question is what the internal debates looked like. Did a security person say, "No, you can't do that; our back-end process isn't robust enough to accept online deposits"? Did the user experience group have to fight with the security group about authentication for account setup? Did the lawyers want to know how well fraudulent transactions could be traced to a particular phone or physical location? Did the head of the security group still try to say, "No, you can't do it; it's just too risky"?

Sometimes, "no" is indeed the right answer. As noted, though, capabilities and environments change. The worst mistake one can make in the computer business is to blithely give yesterday's answer to today's question. The second worst mistake, of course, is rejecting yesterday's answer without thinking about it. The technical and economic constraints may be the same; alternately, the same answer may be correct for an entirely different reason. The challenge is performing the analysis correctly.

1.2 Adapting to Change

There are many ways to deal with change and its likelihood. You can leave enough hooks to handle all possible future contingencies; you can reject changes until you're dragged into the future, kicking and screaming (or go out of business); you can embrace all changes, willy-nilly—or you can stop to do the sober, careful analysis that the problem demands.

Planning for all contingencies is the simplest and most common option. After all, everyone who has been in the business more than a few years *knows* that change will come, and will come in unpredictable ways. There are a number of problems with this approach. For one thing, it's ugly and produces ugly systems. Jon Postel said it well [Comerford 1998]:

> It's perfectly appropriate to be upset. I thought of it in a slightly different way—like a space that we were exploring and, in the early days, we figured out this consistent path through the space: IP, TCP, and so on. What's been happening over the last few years is that the IETF is filling the rest of the space with every alternative approach, not necessarily any better. Every possible alternative is now being written down. And it's not useful.

Planning for everything also produces complex and bloated systems, and while memory and CPU are not critical resources these days, the engineering time to build, maintain, and

Hackers

> Once in ancient days, the then King of England told Sir Christopher Wren, whose name is yet remembered, that the new Cathedral of St. Paul which he had designed was "awful, pompous and artificial." Kings have seldom been noted for perspicacity.
>
> . . .
>
> In the case of the King and Sir Christopher, however, a compliment was intended. A later era would have used the words "awe-inspiring, stately, and ingeniously conceived."
>
> "A Tragedy of Errors"
> —POUL ANDERSON

Words' meanings change over time. Once upon a time, "hacker" might indeed have meant "A person who enjoys exploring the details of programmable systems and how to stretch their capabilities, as opposed to most users, who prefer to learn only the minimum necessary."[a] That isn't the way it is commonly used today. In this book, I'll be using it to mean "A person who uses his skill with computers to try to gain unauthorized access to computer files or networks," per the OED; when writing about security, that is the commonly accepted definition. It is, perhaps, worth noting that the OED traces that usage to 1976, the same year as its first citation for "A person with an enthusiasm for programming or using computers as an end in itself." And if you prefer older meanings, we can go back to either 1481's "That which hacks; an implement for hacking, chopping wood, or breaking up earth; a chopper, cleaver; a hoe, mattock," or 1581's "A 'cutter', cut-throat, bully".

a. "The New Hacker's Dictionary," http://outpost9.com/reference/jargon/jargon_23.html#SEC30.

configure such systems is expensive and becoming more so. From a security perspective, though, complexity is fatal. *No one* understands a complex system, from the architects and programmers who design and build it to the engineers who have to configure it. A 1994 study showed that about 25% of security flaws were due to bugs in the specification, not the code [Landwehr et al. 1994]. In other words, it's not just a programming problem.

Let me give an example of how complexity—necessary complexity, in this case—can lead to a security problem. A web posting [Chan 2011] detailed how an Apple Smart Cover can be used to override the security lock on an Apple iPad 2 running iOS 5.0. (For those who are not initiates into the High Mysteries of the Cult of Apple, a Smart Cover is held to the iPad 2 via magnets. When the cover is peeled back, a sensor inside the iPad 2 notices the absence of the magnet, and wakes up the display. Also, to power off an iPad 2, you hold down the Power button for a few seconds until a confirmation request appears; at that point, you swipe across the designated area of the screen.)

The attack works as follows:

- Lock a passcode-enabled iPad 2.
- Hold down the Power button until it reaches the shutoff slider screen.
- Close the Smart Cover.
- Open the Smart Cover.
- Tap cancel.

What led someone to discover this attack?

If one simply presses the Home button on a blank-screen iPad 2, the lock screen illuminates; if nothing further is done, it blanks again after 10 seconds. On the other hand, if one opens a Smart Cover, the screen remains illuminated for 60 seconds, again reverting to a blank screen if nothing is done. On the gripping hand [Niven and Pournelle 1993], if one initiates the power-down sequence from a blank screen but does nothing, after 30 seconds the device switches to a very dim, non-interactive wallpaper screen. In other words, it goes through a very different sequence of states for each of these ways for waking up the display. The author of the aforementioned posting wrote, "I don't know how anyone would've figured that out but it definitely works."

Consider the state transition diagram shown in Figure 1.1. Two of the states, Covered and Lock screen, are parameterized: the transition from them depends on how they were entered. Arguably, they should be shown as separate nodes on the graph; however, the behavior suggests a single code path with memory. That is the key to the attack.

Someone who thinks like a security person (see Chapter 2) might wonder what would happen if an unexpected transition were to occur. In particular, consider the dotted arc

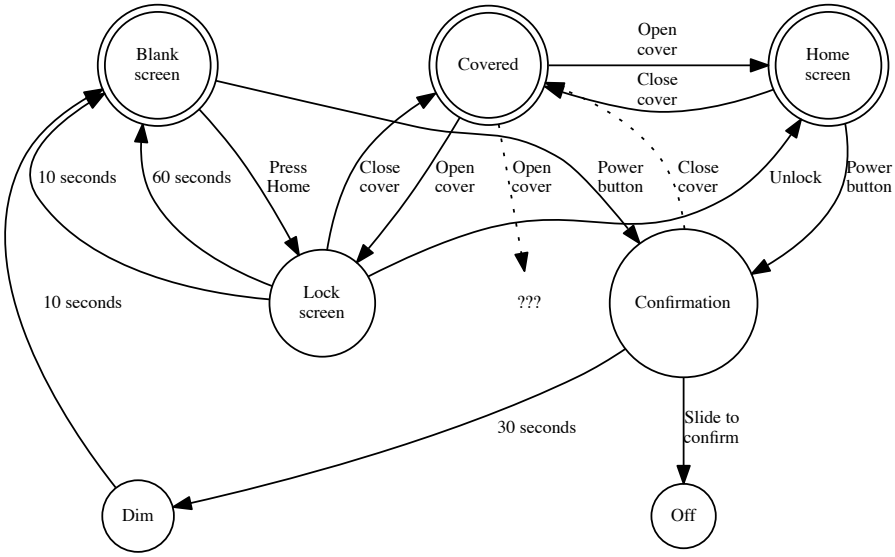

Figure 1.1: Simplified state transitions for unlocking or powering down an iPad 2. Activity can start from the Covered, Home, or Blank states. The dotted lines show the attack. Note that there are three possible destination states if the cover is opened, and two different ways to go from the lock screen to a blank screen.

from the Confirmation state to the Covered state. There is clearly memory in that state, since opening the cover can go to two different places. Is this memory always properly initialized? Clearly not—the actual transition in this case goes to the Home screen state, rather than the Lock screen state.

Change introduces complexity, but the risks of resisting all changes—the second common option—are sufficiently obvious that I won't belabor them, save to recall Ken Olsen's comment that "the personal computer will fall flat on its face in business" [Rifkin 2011]. Olsen was co-founder of the Digital Equipment Corporation, a computer company that no longer exists. Ignoring the world is not a security risk per se; nevertheless, the purpose of computer security is not security for its own sake, but to enable some other operation to function properly and (in some cases) profitably. Let me stress that: the purpose of an organization—a business, a school, a government agency, a hospital—is *not* to be secure; rather, security is an aid to carrying out its real purpose.

The third option for dealing with change is to embrace it. When a new device comes out, start using it. When there's a new service, install it. That way lies disaster; the risks are not always clear. Some years ago, Bruce Schneier wrote about the security risks of Unicode [Schneier 2000], the standard for handling all of the world's alphabets. At the time, I was skeptical, but he was quite correct. The first attack was fairly obvious—since the same glyph in different alphabets (such as the Cyrillic "a" versus the Latin "a") has different codepoints, the domain http://www.pаypal.com/ is different than the domain http://www.paypal.com, but the two look identical on screen [Schneier 2005]. A more subtle attack was discovered recently; it relies on the Unicode metacharacters to cause right-to-left rendering (necessary for languages like Hebrew and Arabic) to hide the `.exe` extension on some files [Krebs 2011b]. Handling cases like these requires not just good code, it requires a good understanding of people's behavior and of the salient characteristics of many different languages.

Note that Unicode or something like it is quite necessary, unless we want to exclude a large fraction of the world's population from the net. That is clearly unacceptable. Nor are the problems a matter of lack of forethought; there was no way, thousands of years ago when some of these languages grew up, that one scribe could say to another, "You know, in the far future, people are going to have these things called computers that will need to handle our alphabet and that of those uncivilized folk across the water, so let's all agree on a direction of writing and on a set of letters." We had to wait for a bit, until the new technology had been deployed and analyzed, for someone to realize that Unicode falls into a classic risk case: two different byte strings can produce the same visual display; the humans who rely on it cannot perceive the internal processing difference.

1.3 Security Analysis

The fourth and best approach to dealing with change is analysis. Naturally, everyone intends to do it, but it isn't easy. Doing it right requires approaching the problem de novo, rather than taking shortcuts. What are the components of the new system? What are their black-box properties? What else do we know or can we guess about them? What are their inputs and outputs? How are things combined? Is every input "secure"? If not, how can it be made secure?

Consider the problem of passwords (a subject that is discussed much more deeply in Chapter 7). Morris and Thompson demonstrated in 1979 that guessable passwords were a security risk [1979]; ever since, "pick strong passwords" has been an entry on every security checklist. However, they were writing about login passwords for a multi-user time-sharing system with remote access. The BIOS password for a server in a physically secure machine is in a very different environment. Do the same rules apply? How about

my home desktop machine? I'm the only user, and while remote logins are possible I've configured the machine so that passwords are not accepted for such logins. Do I need a strong password? (If you're wondering, I have one, because I'm unconvinced that I'll always get the configuration right; if nothing else, a system update from Apple might change that file.)

It is tempting to say "ignore maxims from the past," but that is not quite correct, either. One should certainly examine the assumptions behind them, whether environmental (as in this case), based on the threat, or based on the assets being protected. (Teasing out these assumptions, especially the implicit ones, is one of the hardest things about security.) On the other hand, there is a lot of wisdom behind some of them; one cannot reject them all out of hand. Indeed, one of my touchstones—that complexity leads to insecurity—is nothing more than one of my personal checklist items, based on about 50 years of experience in the field and on a lot of research by many, many people.

Despite the vast and rapid changes in the computer business, the nature of threats hasn't changed much. Certainly, the technical details evolve over time; if nothing else, before there were web servers there was no need to know how to secure one. When web servers—or rifcaghy servers, whatever they are—do exist, though, you need to know those details. More importantly, you need to be able to answer several different questions:

- Do we need a rifcaghy server? More precisely, is there a *business* need for one, where "business" is shorthand for "the purpose of my organization"?
- What are the risks of rifcaghy servers? How can those risks be ameliorated?
- How much confidence do I have in that analysis?
- Are the residual risks more or less than the value to the business of running that service?

Security checklists are not going to give you the answer to any of those questions, until much of the world has been running rifcaghy for quite a while. At that point, you're behind the business curve; perhaps more seriously from a security perspective, your fellow employees may have been using external rifcaghy servers for quite some time, quite unaware of the risks. (Not convinced? Substitute "social media" for "rifcaghy." Substitute "smart phone." Substitute "cloud storage service.")

Furthermore, securing an organization or even a service is not something that's done once. As noted, technologies change, and software is updated. There are new devices, new connectivity, new threats, and new defenses. Even apart from that, designing a security solution is itself an iterative process. The corporate security group or even the security function in a smaller organization generally cannot design a meaningful protective architecture and simply toss it over the wall to the application programmers—and if they

try to do so, the application programmers are likely to take the architecture and toss it themselves. I hesitate to say exactly where, but I do observe that it's very hard to fill up /dev/null.

The problem is that a security solution conceived of by only the security group is likely to pay too little attention to issues like cost and functionality. The problem is not just the cost of the security pieces itself or even the additional development cost; rather, it is the likely loss of functionality or market appeal of the actual product, and it is products that pay everyone's salaries. A perfectly secure service that no one will tolerate using is quite worthless; a security solution of that kind can, will, and should be rejected.

Given all that, the proper process looks like this:

while true **repeat**

1. Identify the assets at risk.
2. Ascertain the enemies interested in each asset, and assess their likely capabilities.
3. Select application technologies.
4. Evaluate the vulnerabilities for each piece.
5. Identify candidate defensive solutions.
6. Estimate the cost, including the cost in damage to the application if security is breached.

It's an iterative process; many solutions involve new assets that themselves need to be protected. Besides, even after you reach a stable answer, the outside world is not standing still; new implementations, new loads, new business requirements, and more mean that the security analysis group will never be out of a job.

But how can this be achieved? When a service is brand new, how is it possible to go through this sort of analysis? Since it would be redundant for me to say that by definition, that's the topic of this book, I'll give a better answer: absent a formal theory of secure system design (and I expect that such a theory will continue to be absent for many years to come), we need to learn from the past. More specifically, by looking at today's security technologies deductively, we can derive appropriate design principles; by applying these inductively, we can reason about tomorrow's rifcaghy services, and even the gushnewly and treltudy services of the day after.

Let's use firewalls as a brief example. (We'll revisit them in much more detail in Chapter 5.) When we look at why they worked in 1994, when Bill Cheswick and I wrote our book on them [1994], we see their basic assumptions: all traffic from the inside would reach the outside Internet via a very small number of chokepoints; only good guys lived on

the inside; and the firewall was capable of filtering out inbound nastiness, operationally defined as anything that didn't match a security policy. How do those principles work today? By and large, they don't. Laptops and smart phones wander across the barrier to the outside, where they're unprotected; malware constitutes an enemy presence inside your organization; policies aren't—can't be—strong enough to describe things like PDF files infected with the latest 0-day exploits. It is tempting to conclude that firewalls are useless.

A deeper look at the principles, though, shows that we can still use firewalls, albeit rather differently. If we can find special circumstances where the principles hold—a server complex is a good example—we can still rely on firewalls as a strong defense. Furthermore, recognition of these principles as explicit guidelines teaches us administrative policies we need to enforce, such as not permitting any mobile devices into the server network, either directly or via a *virtual private network (VPN)*. That in turn says something about the resources we need to give the developers and administrators of that server complex.

This is how we have to proceed in the future. We shouldn't discard the past, nor should we let it straight-jacket us. Rather, we should use it as a guide.

1.4 A Few Words on Terminology

> "You keep using that word. I do not think it means what you think it means."
>
> Inigo in *The Princess Bride*
> —WILLIAM GOLDMAN

I've already explained what I mean by "hacker" and why I use the word (page 6); no more needs to be said about that issue.

Several unusual terms—targetier, APT, Andromedan, and more—are explained in Chapter 3.

There are a number of technical terms I use freely, under the assumption that you know what I mean; these include RSA, MAC address, ARP spoofing, and more. As I noted in the preface, this book is not intended as an introductory text.

Finally, I will often use phrases like "business" or "business purpose." By no means do I intend to imply that this book is limited to the for-profit sector. Although I have indeed worked in industry, I'm now a professor and have served as Chief Technologist of the US Federal Trade Commission. All of these organizations have a goal, whether it's

to make money, instruct students, perform research, protect the nation, or what have you. "Work"—and of course that includes computer work—done on behalf of any of these serves to further those goals. In the interests of clearer writing, I generally use the simpler form to include all of these. Please make mental substitutions as appropriate.