Dr. Dobbs Jolt Award Finalist 2014

Adam Shostack
Microsoft's Threat Modeling Expert

threat modeling

designing for security

WILEY

Part I
Getting Started

This part of the book is for those who are new to threat modeling, and it assumes no prior knowledge of threat modeling or security. It focuses on the key new skills that you'll need to threat model and lays out a methodology that's designed for people who are new to threat modeling.

Part I also introduces the various ways to approach threat modeling using a set of toy analogies. Much like there are many children's toys for modeling, there are many ways to threat model. There are model kits with precisely molded parts to create airplanes or ships. These kits have a high degree of fidelity and a low level of flexibility. There are also numerous building block systems such as Lincoln Logs, Erector Sets, and Lego blocks. Each of these allows for more flexibility, at the price of perhaps not having a propeller that's quite right for the plane you want to model.

In threat modeling, there are techniques that center on attackers, assets, or software, and these are like Lincoln Logs, Erector Sets, and Lego blocks, in that each is powerful and flexible, each has advantages and disadvantages, and it can be tricky to combine them into something beautiful.

Part I contains the following chapters:

- **Chapter 1: Dive In and Threat Model!** contains everything you need to get started threat modeling, and does so by focusing on four questions:
  - What are you building?
  - What can go wrong?

- What should you do about those things that can go wrong?
- Did you do a decent job of analysis?

These questions aren't just what you need to get started, but are at the heart of the four-step framework, which is the core of this book.

- **Chapter 2: Strategies for Threat Modeling** covers a great many ways to approach threat modeling. Many of them are “obvious” approaches, such as thinking about attackers or the assets you want to protect. Each is explained, along with why it works less well than you hope. These and others are contrasted with a focus on software. Software is what you can most reasonably expect a software professional to understand, and so models of software are the most important lesson of Chapter 2. Models of software are one of the two models that you should focus on when threat modeling.

Chapter 1
Dive In and Threat Model!

Anyone can learn to threat model, and what's more, everyone should. Threat modeling is about using models to find security problems. Using a model means abstracting away a lot of details to provide a look at a bigger picture, rather than the code itself. You model because it enables you to find issues in things you haven't built yet, and because it enables you to catch a problem before it starts. Lastly, you threat model as a way to anticipate the threats that could affect you.

Threat modeling is first and foremost a practical discipline, and this chapter is structured to reflect that practicality. Even though this book will provide you with many valuable definitions, theories, philosophies, effective approaches, and well-tested techniques, you'll want those to be grounded in experience. Therefore, this chapter avoids focusing on theory and ignores variations for now and instead gives you a chance to learn by experience.

To use an analogy, when you start playing an instrument, you need to develop muscles and awareness by playing the instrument. It won't sound great at the start, and it will be frustrating at times, but as you do it, you'll find it gets easier. You'll start to hit the notes and the timing. Similarly, if you use the simple four-step breakdown of how to threat model that's exercised in Parts I-III of this book, you'll start to develop your muscles. You probably know the old joke about the person who stops a musician on the streets of New York and asks “How do I get to Carnegie Hall?” The answer, of course, is “practice, practice, practice.” Some of that includes following along, doing the exercises, and

developing an understanding of the steps involved. As you do so, you'll start to understand how the various tasks and techniques that make up threat modeling come together.

In this chapter you're going to find security flaws that might exist in a design, so you can address them. You'll learn how to do this by examining a simple web application with a database back end. This will give you an idea of what can go wrong, how to address it, and how to check your work. Along the way, you'll learn to play *Elevation of Privilege*, a serious game designed to help you start threat modeling. Finally you'll get some hands-on experience building your own threat model, and the chapter closes with a set of checklists that help you get started threat modeling.

Learning to Threat Model

You begin threat modeling by focusing on four key questions:

1. What are you building?
2. What can go wrong?
3. What should you do about those things that can go wrong?
4. Did you do a decent job of analysis?

In addressing these questions, you start and end with tasks that all technologists should be familiar with: drawing on a whiteboard and managing bugs. In between, this chapter will introduce a variety of new techniques you can use to think about threats. If you get confused, just come back to these four questions.

Everything in this chapter is designed to help you answer one of these questions. You're going to first walk through

these questions using a three-tier web app as an example, and after you've read that, you should walk through the steps again with something of your own to threat model. It could be software you're building or deploying, or software you're considering acquiring. If you're feeling uncertain about what to model, you can use one of the sample systems in this chapter or an exercise found in Appendix E, “Case Studies.”

The second time you work through this chapter, you'll need a copy of the *Elevation of Privilege* threat-modeling game. The game uses a deck of cards that you can download free from `threatmodelingbook.com/resources`. You should get two–four friends or colleagues together for the game part.

You start with building a diagram, which is the first of four major activities involved in threat modeling and is explained in the next section. The other three include finding threats, addressing them, and then checking your work.

What Are You Building?

Diagrams are a good way to communicate what you are building. There are lots of ways to diagram software, and you can start with a whiteboard diagram of how data flows through the system. In this example, you're working with a simple web app with a web browser, web server, some business logic and a database (see Figure 1.1).

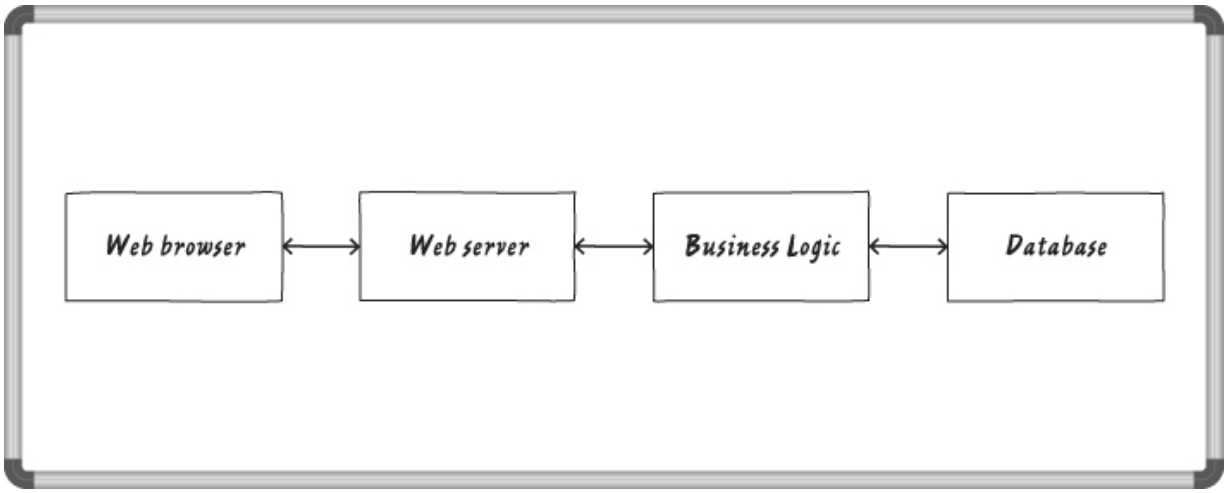

Figure 1.1 A whiteboard diagram

Some people will actually start thinking about what goes wrong right here. For example, how do you know that the web browser is being used by the person you expect? What happens if someone modifies data in the database? Is it OK for information to move from one box to the next without being encrypted? You might want to take a minute to think about some things that could go wrong here because these sorts of questions may lead you to ask “is that allowed?” You can create an even better model of what you're building if you think about “who controls what” a little. Is this a website for the whole Internet, or is it an intranet site? Is the database on site, or at a web provider?

For this example, let's say that you're building an Internet site, and you're using the fictitious Acme storage-system. (I'd put a specific product here, but then I'd get some little detail wrong and someone, certainly not you, would get all wrapped around the axle about it and miss the threat modeling lesson. Therefore, let's just call it Acme, and pretend it just works the way I'm saying. Thanks! I knew you'd understand.)

Adding boundaries to show who controls what is a simple way to improve the diagram. You can pretty easily see that

the threats that cross those boundaries are likely important ones, and may be a good place to start identifying threats. These boundaries are called *trust boundaries*, and you should draw them wherever different people control different things. Good examples of this include the following:

- Accounts (UIDs on unix systems, or SIDS on Windows)
- Network interfaces
- Different physical computers
- Virtual machines
- Organizational boundaries
- Almost anywhere you can argue for different privileges

Trust Boundary versus Attack Surface

A closely related concept that you may have encountered is *attack surface*. For example, the hull of a ship is an attack surface for a torpedo. The side of a ship presents a larger attack surface to a submarine than the bow of the same ship. The ship may have internal “trust” boundaries, such as waterproof bulkheads or a Captain's safe. A system that exposes lots of interfaces presents a larger attack surface than one that presents few APIs or other interfaces. Network firewalls are useful boundaries because they reduce the attack surface relative to an external attacker. However, much like the Captain's safe, there are still trust boundaries inside the firewall. A trust boundary and an attack surface are very similar views of the same thing. An attack surface is a trust boundary and a direction from which an attacker could launch an attack. Many people will treat the terms are interchangeable. In this book, you'll generally see “trust boundary” used.

In your diagram, draw the trust boundaries as boxes (see Figure 1.2), showing what's inside each with a label (such as “corporate data center”) near the edge of the box.

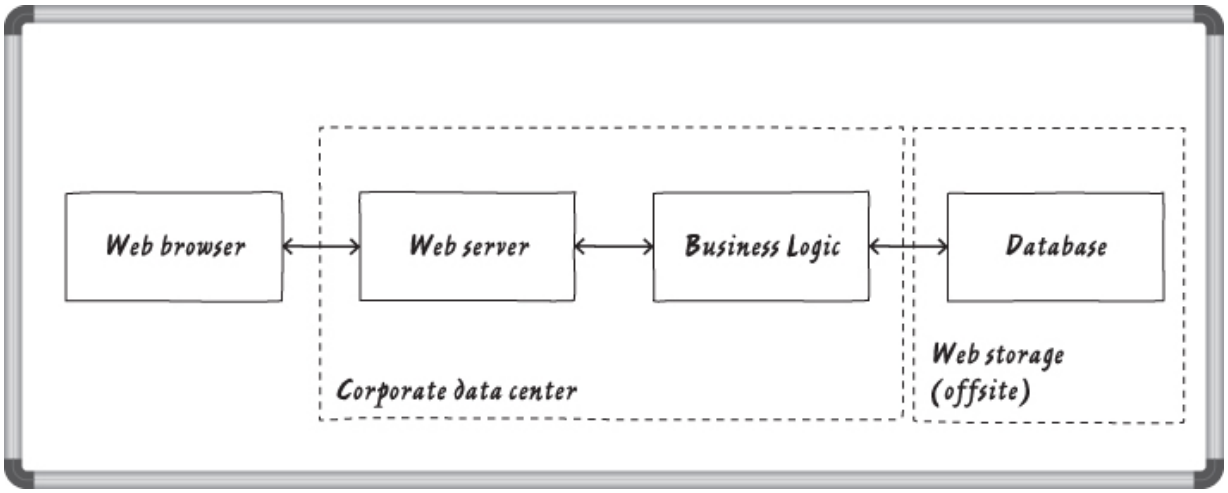

Figure 1.2 Trust boundaries added to a whiteboard diagram

As your diagram gets larger and more complex, it becomes easy to miss a part of it, or to become confused by labels on the data flows. Therefore, it can be very helpful to number each process, data flow, and data store in the diagram, as shown in Figure 1.3. (Because each trust boundary should have a unique name, representing the unique trust inside of it, there's limited value to numbering those.)

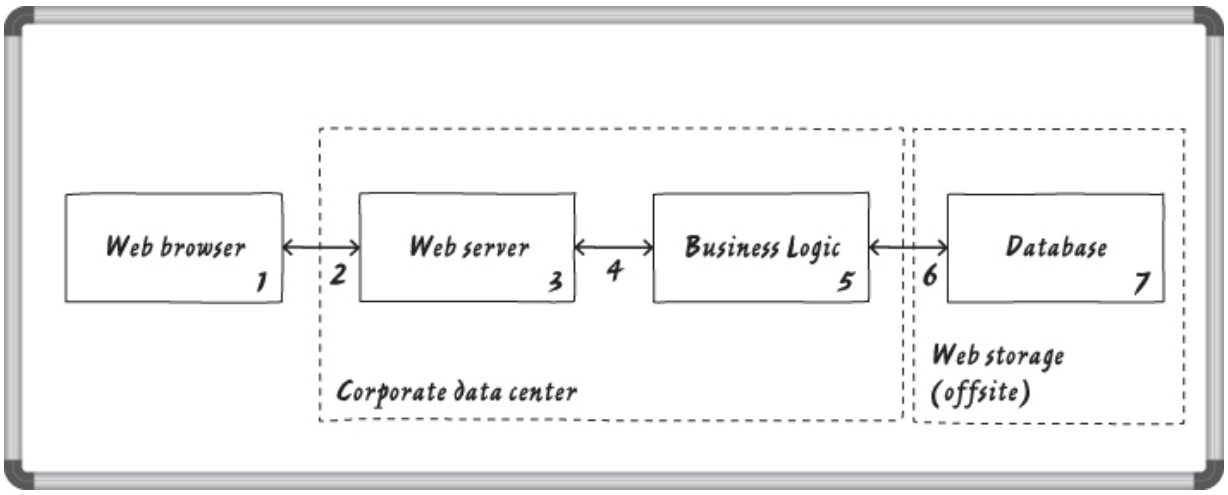

Figure 1.3 Numbers and trust boundaries added to a whiteboard diagram

Regarding the physical form of the diagram: Use whatever works for you. If that's a whiteboard diagram and a camera

phone picture, great. If it's Visio, or OmniGraffle, or some other drawing program, great. You should think of threat model diagrams as part of the development process, so try to keep it in source control with everything else.

Now that you have a diagram, it's natural to ask, is it the right diagram? For now, there's a simple answer: Let's assume it is. Later in this chapter there are some tips and checklists as well as a section on updating the diagram, but at this stage you have a good enough diagram to get started on identifying threats, which is really why you bought this book. So let's identify.

What Can Go Wrong?

Now that you have a diagram, you can really start looking for what can go wrong with its security. This is so much fun that I turned it into a game called, *Elevation of Privilege*. There's more on the game in Appendix D, “Elevation of Privilege: The Cards,” which discusses each card, and in Chapter 11, “Threat Modeling Tools,” which covers the history and philosophy of the game, but you can get started playing now with a few simple instructions. If you haven't already done so, download a deck of cards from http://www.microsoft.com/security/sdl/adopt/eop.aspx. Print the pages in color, and cut them into individual cards. Then shuffle the deck and deal it out to those friends you've invited to play.

> **Note**
>
> Some people aren't used to playing games at work. Others approach new games with trepidation, especially when those games involve long, complicated instructions. *Elevation of Privilege* takes just a few lines to explain. You should give it a try.

How To Play Elevation of Privilege

Elevation of Privilege is a serious game designed to help you threat model. A sample card is shown in Figure 1.4. You'll notice that like playing cards, it has a number and suit in the upper left, and an example of a threat as the main text on the card. To play the game, simply follow the instructions in the upcoming list.

1. Deal the deck. (Shuffling is optional.)
2. The person with the 3 of Tampering leads the first round. (In card games like this, rounds are also called “tricks” or “hands.”)
3. Each round works like so:
   A. Each player plays one card, starting with the person leading the round, and then moving clockwise.
   B. To play a card, read it aloud, and try to determine if it affects the system you have diagrammed. If you can link it, write it down, and score yourself a point. Play continues clockwise with the next player.
   C. When each player has played a card, the player who has played the highest card wins the round. That player leads the next round.

4. When all the cards have been played, the game ends and the person with the most points wins.
5. If you're threat modeling a system you're building, then you go file any bugs you find.

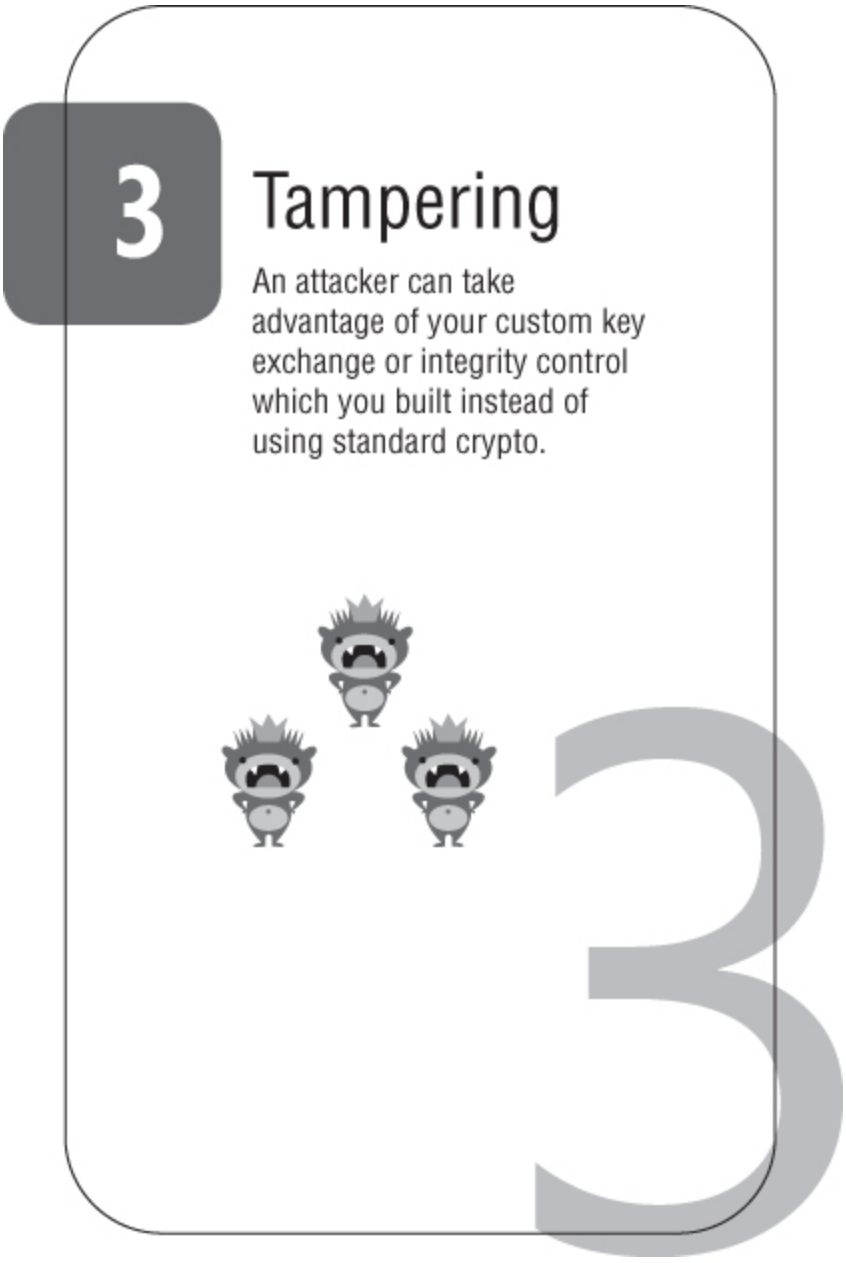

Figure 1.4 An *Elevation of Privilege* card

There are some folks who threat model like this in their sleep, or even have trouble switching it off. Not everyone is

like that. That's OK. Threat modeling is not rocket science. It's stuff that anyone who participates in software development can learn. Not everyone wants to dedicate the time to learn to do it in their sleep.

Identifying threats can seem intimidating to a lot of people. If you're one of them, don't worry. This section is designed to gently walk you through threat identification. Remember to have fun as you do this. As one reviewer said: “Playing *Elevation of Privilege* should be *fun*. Don't downplay that. We play it every Friday. It's enjoyable, relaxing, and still has business value.”

Outside of the context of the game, you can take the next step in threat modeling by thinking of things that might go wrong. For instance, how do you know that the web browser is being used by the person you expect? What happens if someone modifies data in the database? Is it OK for information to move from one box to the next without being encrypted? You don't need to come up with these questions by just staring at the diagram and scratching your chin. (I didn't!) You can identify threats like these using the simple mnemonic STRIDE, described in detail in the next section.

Using the STRIDE Mnemonic to Find Threats

STRIDE is a mnemonic for things that go wrong in security. It stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege:

- **Spoofing** is pretending to be something or someone you're not.
- **Tampering** is modifying something you're not supposed to modify. It can include packets on the wire (or wireless), bits on disk, or the bits in memory.

- **Repudiation** means claiming you didn't do something (regardless of whether you did or not).
- **Information Disclosure** is about exposing information to people who are not authorized to see it.
- **Denial of Service** are attacks designed to prevent a system from providing service, including by crashing it, making it unusably slow, or filling all its storage.
- **Elevation of Privilege** is when a program or user is technically able to do things that they're not supposed to do.

> ## Note
>
> This is where *Elevation of Privilege*, the game, gets its name. This book uses *Elevation of Privilege*, italicized, or abbreviated to EoP, for the game—to avoid confusion with the threat.

Recall the three example threats mentioned in the preceding section:

- How do you know that the web browser is being used by the person you expect?
- What happens if someone modifies data in the database?
- Is it ok for information to go from one box to the next without being encrypted?

These are examples of spoofing, tampering, and information disclosure. Using STRIDE as a mnemonic can help you walk through a diagram and select example threats. Pair that with a little knowledge of security and the right techniques, and you'll find the important threats

faster and more reliably. If you have a process in place for ensuring that you develop a threat model, document it, and you can increase confidence in your software.

Now that you have STRIDE in your tool belt, walk through your diagram again and look for more threats, this time using the mnemonic. Make a list as you go with the threat and what element of the diagram it affects. (Generally, the software, data flow, or storage is affected, rather than the trust boundary.) The following list provides some examples of each threat.

- **Spoofing:** Someone might pretend to be another customer, so you'll need a way to authenticate users. Someone might also pretend to be your website, so you should ensure that you have an SSL certificate and that you use a single domain for all your pages (to help that subset of customers who read URLs to see if they're in the right place). Someone might also place a deep link to one of your pages, such as `logout.html` or `placeorder.aspx`. You should be checking the Referrer field before taking action. That's not a complete solution to what are called CSRF (Cross Site Request Forgery) attacks, but it's a start.
- **Tampering:** Someone might tamper with the data in your back end at Acme. Someone might tamper with the data as it flows back and forth between their data center and yours. A programmer might replace the operational code on the web front end without testing it, thinking they're uploading it to staging. An angry programmer might add a coupon code “PayBobMore” that offers a 20 percent discount on all goods sold.
- **Repudiation:** Any of the preceding actions might require digging into what happened. Are there system

logs? Is the right information being logged effectively? Are the logs protected against tampering?

- **Information Disclosure:** What happens if Acme reads your database? Can anyone connect to the database and read or write information?
- **Denial of Service:** What happens if a thousand customers show up at once at the website? What if Acme goes down?
- **Elevation of Privilege:** Perhaps the web front end is the only place customers should access, but what enforces that? What prevents them from connecting directly to the business logic server, or uploading new code? If there's a firewall in place, is it correctly configured? What controls access to your database at Acme, or what happens if an employee at Acme makes a mistake, or even wants to edit your files?

The preceding possibilities aren't intended to be a complete list of how each threat might manifest against every model. You can find a more complete list in Chapter 3, “STRIDE.” This shorter version will get you started though, and it is focused on what you might need to investigate based on the very simple diagram shown in Figure 1.2. Remember the musical instrument analogy. If you try to start playing the piano with Ravel's Gaspard (regarded as one of the most complex piano pieces ever written), you're going to be frustrated.

Tips for Identifying Threats

Whether you are identifying threats using *Elevation of Privilege*, STRIDE, or both, here are a few tips to keep in mind that can help you stay on the right track to determine what could go wrong:

- **Start with external entities:** If you're not sure where to start, start with the external entities or events which drive activity. There are many other valid approaches though: You might start with the web browser, looking for spoofing, then tampering, and so on. You could also start with the business logic if perhaps your lead developer for that component is in the room. Wherever you choose to begin, you want to aspire to some level of organization. You could also go in “STRIDE order” through the diagram. Without some organization, it's hard to tell when you're done, but be careful not to add so much structure that you stifle creativity.
- **Never ignore a threat because it's not what you're looking for right now:** You might come up with some threats while looking at other categories. Write them down and come back to them. For example, you might have thought about “can anyone connect to our database,” which is listed under information disclosure, while you were looking for spoofing threats. If so, that's awesome! Good job! Redundancy in what you find can be tedious, but it helps you avoid missing things. If you find yourself asking whether “someone not authorized to connect to the database who reads information” constitutes spoofing or information disclosure, the answer is, who cares? Record the issue and move along to the next one. STRIDE is a tool to guide you to threats, not to ask you to categorize what you've found; it makes a lousy taxonomy, anyway. (That is to say, there are plenty of security issues for which you can make an argument for various different categorizations. Compare and contrast it with a good taxonomy, such as the taxonomy of life. Does it have a backbone? If so, it's a vertebrate.)
- **Focus on feasible threats:** Along the way, you might come up with threats like “someone might insert a back

door at the chip factory," or "someone might hire our janitorial staff to plug in a hardware key logger and steal all our passwords." These are real possibilities but not very likely compared to using an exploit to attack a vulnerability for which you haven't applied the patch, or tricking someone into installing software. There's also the question of what you can do about either, which brings us to the next section.

Addressing Each Threat

You should now have a decent-sized list or lists of threats. The next step in the threat modeling process is to go through the lists and address each threat. There are four types of action you can take against each threat: Mitigate it, eliminate it, transfer it, or accept it. The following list looks briefly at each of these ways to address threats, and then in the subsequent sections you will learn how to address each specific threat identified with the STRIDE list in the "What Can Go Wrong" section. For more details about each of the strategies and techniques to address these threats, see Chapters 8 and 9, "Defensive Building Blocks" and "Tradeoffs When Addressing Threats."

- **Mitigating threats** is about doing things to make it harder to take advantage of a threat. Requiring passwords to control who can log in mitigates the threat of spoofing. Adding password controls that enforce complexity or expiration makes it less likely that a password will be guessed or usable if stolen.
- **Eliminating threats** is almost always achieved by eliminating features. If you have a threat that someone will access the administrative function of a website by visiting the `/admin/URL`, you can mitigate it with passwords or other authentication techniques, but the threat is still present. You can make it less likely to be found by using

a URL like `/j8e8vg21euwq/`, but the threat is still present. You can eliminate it by removing the interface, handling administration through the command line. (There are still threats associated with how people log in on a command line. Moving away from HTTP makes the threat easier to mitigate by controlling the attack surface. Both threats would be found in a complete threat model.) Incidentally, there are other ways to eliminate threats if you're a mob boss or you run a police state, but I don't advocate their use.

- **Transferring threats** is about letting someone or something else handle the risk. For example, you could pass authentication threats to the operating system, or trust boundary enforcement to a firewall product. You can also transfer risk to customers, for example, by asking them to click through lots of hard-to-understand dialogs before they can do the work they need to do. That's obviously not a great solution, but sometimes people have knowledge that they can contribute to making a security tradeoff. For example, they might know that they just connected to a coffee shop wireless network. If you believe the person has essential knowledge to contribute, you should work to help her bring it to the decision. There's more on doing that in Chapter 15, “Human Factors and Usability.”
- **Accepting the risk** is the final approach to addressing threats. For most organizations most of the time, searching everyone on the way in and out of the building is not worth the expense or the cost to the dignity and job satisfaction of those workers. (However, diamond mines and sometimes government agencies take a different approach.) Similarly, the cost of preventing someone from inserting a back door in the motherboard is expensive, so for each of these examples you might choose to accept the risk. And once you've accepted the

risk, you shouldn't worry over it. Sometimes worry is a sign that the risk hasn't been fully accepted, or that the risk acceptance was inappropriate.

The strategies listed in the following tables are intended to serve as examples to illustrate ways to address threats. Your “go-to” approach should be to mitigate threats. Mitigation is generally the easiest and the best for your customers. (It might look like accepting risk is easier, but over time, mitigation is easier.) Mitigating threats can be hard work, and you shouldn't take these examples as complete. There are often other valid ways to address each of these threats, and sometimes trade-offs must be made in the way the threats are addressed.

Addressing Spoofing

Table 1.1 and the list that follows show targets of spoofing, mitigation strategies that address spoofing, and techniques to implement those mitigations.

- When you're concerned about a person being spoofed, ensure that each person has a unique username and some way of authenticating. The traditional way to do this is with passwords, which have all sorts of problems as well as all sorts of advantages that are hard to replicate. See Chapter 14, “Accounts and Identity” for more on passwords.
- When accessing a file on disk, don't ask for the file with `open(file)`. Use `open(/path/to/file)`. If the file is sensitive, after opening, check various security elements of the file descriptor (such as fully resolved name, permissions, and owner). You want to check with the file descriptor to avoid *race conditions*. This applies doubly when the file is an executable, although checking after opening can be tricky. Therefore, it may help to ensure that the

permissions on the executable can't be changed by an attacker. In any case, you almost never want to call `exec()` with `./file`.

- When you're concerned about a system or computer being spoofed when it connects over a network, you'll want to use DNSSEC, SSL, IPsec, or a combination of those to ensure you're connecting to the right place.

Table 1.1 Addressing Spoofing Threats

Threat Target	Mitigation Strategy	Mitigation Technique
Spoofing a person	Identification and authentication (usernames and something you know/have/are)	Usernames, real names, or other identifiers: ▪ Passwords ▪ Tokens ▪ Biometrics Enrollment/maintenance/expiry
Spoofing a “file” on disk	Leverage the OS	▪ Full paths ▪ Checking ACLs ▪ Ensuring that pipes are created properly
	Cryptographic authenticators	Digital signatures or authenticators
Spoofing a network address	Cryptographic	▪ DNSSEC ▪ HTTPS/SSL ▪ IPsec
Spoofing a program in memory	Leverage the OS	Many modern operating systems have some form of application identifier that the OS will enforce.

Addressing Tampering

Table 1.2 and the list that follows show targets of tampering, mitigation strategies that address tampering, and techniques to implement those mitigations.

- **Tampering with a file:** Tampering with files can be easy if the attacker has an account on the same machine, or by tampering with the network when the files are obtained from a server.
- **Tampering with memory:** The threats you want to worry about are those that can occur when a process with less privileges than you, or that you don't trust, can alter memory. For example, if you're getting data from a shared memory segment, is it ACLed so only the other process can see it? For a web app that has data coming in via AJAX, make sure you validate that the data is what you expect after you pull in the right amount.
- **Tampering with network data:** Preventing tampering with network data requires dealing with both spoofing and tampering. Otherwise, someone who wants to tamper can simply pretend to be the other end, using what's called a *man-in-the-middle attack*. The most common solution to these problems is SSL, with IP Security (IPsec) emerging as another possibility. SSL and IPsec both address confidentiality and tampering, and can help address spoofing.
- **Tampering with networks anti-pattern:** It's somewhat common for people to hope that they can isolate their network, and so not worry about tampering threats. It's also very hard to maintain isolation over time. Isolation doesn't work as well as you would hope. For example, the isolated United States SIPRNet was thoroughly infested with malware, and the operation to clean it up took 14 months (Shachtman, 2010).

Note

A program can't check whether it's authentic after it loads. It may be possible for something to rely on “trusted bootloaders” to provide a chain of signatures, but the security decisions are being made external to that code. (If you're not familiar with the technology, don't worry, the key lesson is that a program cannot check its own authenticity.)

Table 1.2 Addressing Tampering Threats

Threat Target	Mitigation Strategy	Mitigation Technique
Tampering with a file	Operating system	ACLs
	Cryptographic	▪ Digital Signatures ▪ Keyed MAC
Racing to create a file (tampering with the file system)	Using a directory that's protected from arbitrary user tampering	ACLs Using private directory structures (Randomizing your file names just makes it annoying to execute the attack.)
Tampering with a network packet	Cryptographic	▪ HTTPS/SSL ▪ IPsec
	Anti-pattern	Network isolation (See note on network isolation anti-pattern.)

Addressing Repudiation

Addressing repudiation is generally a matter of ensuring that your system is designed to log and ensuring that those logs are preserved and protected. Some of that can be handled with simple steps such as using a reliable

transport for logs. In this sense, syslog over UDP was almost always silly from a security perspective; syslog over TCP/SSL is now available and is vastly better.

Table 1.3 and the list that follows show targets of repudiation, mitigation strategies that address repudiation, and techniques to implement those mitigations.

- **No logs means you can't prove anything:** This is self-explanatory. For example, when a customer calls to complain that they never got their order, how will this be resolved? Maintain logs so that you can investigate what happens when someone attempts to repudiate something.
- **Logs come under attack:** Attackers will do things to prevent your logs from being useful, including filling up the log to make it hard to find the attack or forcing logs to "roll over." They may also do things to set off so many alarms that the real attack is lost in a sea of troubles. Perhaps obviously, sending logs over a network exposes them to other threats that you'll need to handle.
- **Logs as a channel for attack:** By design, you're collecting data from sources outside your control, and delivering that data to people and systems with security privileges. An example of such an attack might be sending mail addressed to "`</html>` haha@example.com", causing trouble for web-based tools that don't expect inline HTML.

Table 1.3 Addressing Repudiation Threats

Threat Target	Mitigation Strategy	Mitigation Technique
No logs means you can't prove anything.	Log	Be sure to log all the security-relevant information.
Logs come under attack	Protect your logs.	▪ Send over the network. ▪ ACL
Logs as a channel for attack	Tightly specified logs	Documenting log design early in the development process

You can make it easier to write secure code to process your logs by clearly communicating what your logs can't contain, such as “Our logs are all plaintext, and attackers can insert all sorts of things,” or “Fields 1–5 of our logs are tightly controlled by our software, fields 6–9 are easy to inject data into. Field 1 is time in GMT. Fields 2 and 3 are IP addresses (v4 or 6)…” Unless you have incredibly strict control, documenting what your logs can contain will likely miss things. (For example, can your logs contain Unicode double-wide characters?)

Addressing Information Disclosure

Table 1.4 and the list which follows show targets of information disclosure, mitigation strategies that address information disclosure, and techniques to implement those mitigations.

- **Network monitoring:** Network monitoring takes advantage of the architecture of most networks to monitor traffic. (In particular, most networks now

broadcast packets, and each listener is expected to decide if the packet matters to them.) When networks are architected differently, there are a variety of techniques to draw traffic to or through the monitoring station.

If you don't address spoofing, much like tampering, an attacker can just sit in the middle and spoof each end. Mitigating network information disclosure threats requires handling both spoofing and tampering threats. If you don't address tampering, then there are all sorts of clever ways to get information out. Here again, SSL and IP Security options are your simplest choices.

- **Names reveal information:** When the name of a directory or a filename itself will reveal information, then the best way to protect it is to create a parent directory with an innocuous name and use operating system ACLs or permissions.
- **File content is sensitive:** When the contents of the file need protection, use ACLs or cryptography. If you want to protect all the data should the machine fall into unauthorized hands, you'll need to use cryptography. The forms of cryptography that require the person to manually enter a key or passphrase are more secure and less convenient. There's file, filesystem, and database cryptography, depending on what you need to protect.
- **APIs reveal information:** When designing an API, or otherwise passing information over a trust boundary, select carefully what information you disclose. You should assume that the information you provide will be passed on to others, so be selective about what you provide. For example, website errors that reveal the username and password to a database are a common form of this flaw, others are discussed in Chapter 3.

Table 1.4 Addressing Information Disclosure Threats

Threat Target	Mitigation Strategy	Mitigation Technique
Network monitoring	Encryption	▪ HTTPS/SSL ▪ IPsec
Directory or filename (for example `layoff-letters/adamshostack.docx`)	Leverage the OS.	ACLs
File contents	Leverage the OS.	ACLS
	Cryptography	File encryption such as PGP, disk encryption (FileVault, BitLocker)
API information disclosure	Design	Careful design control Consider pass by reference or value.

Addressing Denial of Service

Table 1.5 and the list that follows show targets of denial of service, mitigation strategies that address denial of service, and techniques to implement those mitigations.