Dr. Dobbs Jolt Award Finalist 2014

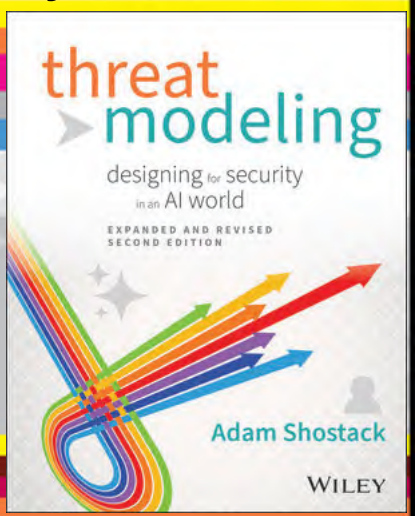
Adam Shostack
Microsoft's Threat Modeling Expert

Chapter 4 "Attack Trees" from

threat modeling

designing for security

New Edition Coming January 2027



WILEY

Threat Modeling: Designing for Security

Published by
John Wiley & Sons, Inc.
10475 Crosspoint
BoulevardIndianapolis, IN 46256
www.wiley.com

Published by John Wiley & Sons, Inc., Indianapolis, Indiana
Published simultaneously in Canada

ISBN: 978-1-118-80999-0
ISBN: 978-1-118-82269-2 (ebk)
ISBN: 978-1-118-81005-7 (ebk)

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2013954095

This excerpt from the first edition of *Threat Modeling: Designing for Security* was removed in favor of a different framing, but this other framing might be helpful.

Starting the Threat Modeling Project

The basic approach of "draw a diagram and use the *Elevation of Privilege* game to find threats" is functional, but people prefer different amounts of prescriptiveness, so this section provides some additional structure that may help you get started.

When to Threat Model

You can threat model at various times during a process, with each choice having a different value. Most important, you should threat model as you get started on a project. The act of drawing trust boundaries early on can greatly help you improve your architecture. You can also threat model as you work through features. This allows you to have smaller, more focused threat modeling projects, keeping your skills sharp and reducing the chance that you'll find big problems at the end. It is also a good idea to revisit the threat model as you get ready to deliver, to ensure that you haven't made decisions which accidentally altered the reality underlying the model.

Starting at the Beginning

Threat modeling as you get started involves modeling the system you're planning or building, finding threats against the model, and filing bugs that you'll track and manage, such as other issues discovered throughout the development process. Some of those bugs may be test cases, some might be feature work, and some may be deployment decisions. It depends on what you're threat modeling.

Working through Features

As you develop each feature, there may be a small amount of threat modeling work to do. That work involves looking deeply at the threats to that feature (and possibly refreshing or validating your understanding of the context by checking the software model). As you start work on a feature or component, it can also be a good time to work through second- or third-order threats. These are the threats in which an attacker will try to bypass the features or design elements that you put in place to block the most immediate threats. For example, if the primary

threat is a car thief breaking a window, a secondary threat is them jumping the ignition. You can mitigate that with a steering-wheel lock, which is thus a second-order mitigation. There's more on this concept of ordered threats in the "Digging Deeper into Mitigations" section later in this chapter, as well as more on considering planned mitigations, and how an attacker might work around them.

Threat modeling as you work through a feature has several important value propositions. One is that if you do a small threat model as you start a component or feature, the design is probably closer to mind. In other words, you'll have a more detailed model with which to look for threats. Another is that if you find threats, they are closer to mind as you're working on that feature. Threat modeling as you work through features can also help you maintain your awareness of threats and your skills at threat modeling. (This is especially true if your project is a long one.)

Close to Delivery

Lastly, you should threat model as you get ready to ship by reexamining the model and checking your bugs. (Shipping here is inclusive of delivering, deploying, or going live.) Reexamining the model means ensuring that everyone still agrees it's a decent model of what you're building, and that it includes all the trust boundaries and data flows that cross them. Checking your bugs involves checking each bug that's tagged threat modeling (or however else you're tracking them), and ensuring it didn't slip through the cracks.

Time Management

So how long should all this take? The answer to that varies according to system size and complexity, the familiarity of the participants with the system, their skill in threat modeling, and even the culture of meetings in an organization. Some very rough rules of thumb are that you should be able to diagram and find threats against a "component" and decide if you need to do more enumeration in a one-hour session with an experienced threat modeler to moderate or help. For the sort of system that a small start-up might build, the end-to-end threat modeling could take a few hours to a week, or possibly longer if the data the system holds is particularly sensitive. At the larger end of the spectrum, a project to diagram the data flows of a large online service has been known to require four people working for several months. That level of effort was required to help find threat variations and alternate routes through a system that had grown to serve millions of people.

Whatever you're modeling, familiarity with threat modeling helps. If you need to refer back to this book every few minutes, your progress will be slower. One of the reasons to threat model regularly is to build skill and familiarity with the tasks and techniques. Organizational culture also plays a part. Organizations

that run meetings with nowhere to sit will likely create a list of threats faster than a consensus-oriented organization that encourages exploring ideas. (Which list will be better is a separate and fascinating question.)

What to Start and (Plan to) End With

When you start looking for threats, a diagram is something between useful and essential input. Experienced modelers may be able to start without it, but will likely iterate through creating a diagram as they find threats. The diagram is likely to change as you use it to find threats; you'll discover things you missed or don't need. That's normal, and unless you run a strict waterfall approach to engineering, it's a process that evolves much like the way requirements evolve as you discover what's easy or hard to build.

Use the following two testable states to help assess when you're done:

- You have filed bugs.
- You have a diagram or diagrams that everyone agrees represents the system.

To be more specific, you should probably have a number of bugs that's roughly scaled to the number of things in your diagram. If you're using a data flow diagram and STRIDE, expect to have about five threats per diagram element.

type="note"

Originally, this text suggested that you should have: (# of processes * 6) + (# of data flows * 3) + (# of data stores * 3.5) + (# of distinct external entities *2) threats, but that requires keeping four separate counts, and is thus more work to get approximately the same answer.

You might notice that says "STRIDE" rather than "STRIDE-per-element" or "STRIDE-per-interaction," and five turns out to match the number you get if you tally up the checkmarks in those charts. That's because those charts are derived from where the threats usually show up.

Where to Start

When you are staring at a blank whiteboard and wondering where to start, there are several commonly recommended places. Many people have recommended assets or attackers, but as you learned in Chapter 2, "Strategies for Threat Modeling," the best starting place is a diagram that covers the system as a whole, and from there start looking at the trust boundaries. For advice on how to create diagrams, see Chapter 2.

When you assemble a group of people in a room to look for threats, you should include people who know about the software, the data flows and (if possible) threat modeling. Begin the process with the component(s) on which the participants in the room are working. You'll want to start top-down, and then work across the system, going "breadth first," rather than delving deep into any component ("depth first").

Finding Threats Top Down

Almost any system should be modeled from the highest-level view you can build of the entire system, for some appropriate value of "the entire system". What constitutes an entire system is, of course, up for debate, just like what constitutes the entire Internet, the entirety of (say) Amazon's website, and so on, isn't a simple question. In such cases more scoping is needed. The ideal is probably what is within an organization's control and, to the extent possible, cross-reviews with those responsible for other components.

In contrast, bottom-up threat modeling starts from features, and then attempts to derive a coherent model from those feature-level models. This doesn't work well, but advice that implies you should do this is common, so a bit of discussion may be helpful. The reason this doesn't work is because it turns out to be very challenging to bring threat models together when they are not derived from a system-level view. As such, you should start from the highest-level view you can build of the entire system.

type="general"

Microsoft's Bottom-up Experience

It may help to understand the sorts of issues that can lead to a bottom-up approach. At Microsoft in the mid-2000s, there was an explosion of bottom-up threat modeling. There were three drivers for this: specific words in the Security Development Lifecycle (SDL) threat model requirement, aspects of Microsoft's approach to function teams, and the work involved in creating top-level models. The SDL required "all new features" be threat modeled. This intersected with an approach to features whereby a particular team of developer, tester, and program manager owns a feature and collaborates to ship it. Because the team owned its feature, it was natural to ask it to add threat models to the specifications involved in producing it. As Microsoft's approach to security evolved, product security teams had diverse sets of important tasks to undertake. Creating all-up threat models was usually not near the top of the list. (Many large product diagrams have now done that work and found it worthwhile. Some of these diagrams require more than one poster-size sheet of paper.)

Finding Threats "Across"

Even with a top-down approach, you want to go breadth first, and there are three different lists you can iterate "across": A list of the trust boundaries, a list of diagram elements, or a list of threats. A structure can help you look for threats, either because you and your team like structure, or because the task feels intimidating and you want to break it down, Table 7-1 shows three approaches.

Table 7-1: Lists to Iterate Across

Method	Sample Statement	Comments
Start from what goes across trust boundaries.	"What can go wrong as foo comes across this trust boundary?	This is likely to identify the highest-value threats.
Iterate across diagram elements.	"What can go wrong with this database file?" "What can go wrong with the logs?"	Focusing on diagram elements may work well when a lot of teams are collaborating.
Iterate across the threats.	"Where are the spoofing threats in this diagram?" "Where are the tampering threats?"	Making threats the focus of discussion may help you find related threats.

Each of these approaches can be a fine way to start, as long as you don't let them become straightjackets. If you don't have a preference, try starting from what crosses trust boundaries, as threats tend to cluster there. However you iterate, ensure that you capture each threat as it comes up, regardless of the planned approach.