

How to Do Threat Modeling A Beginner StepbyStep Walkthrough

To perform threat modeling, (1) draw the system as a simple data flow diagram, (2) mark every trust boundary, (3) ask what could go wrong at each boundary and component, (4) rank the threats by likelihood and impact, (5) decide on a control for each serious threat and (6) check your work and repeat when the design changes. You can complete a useful first model in about an hour with a whiteboard and a small team.

Most people meet threat modelling the same way someone says we should threat model this and everyone nods, but nobody is quite sure where to begin. The term sounds academic. The practice is not. At its core, it is a structured conversation about how a system could be attacked, held while the design can still be changed cheaply.

This guide is written for people doing it for the first time. There are no frameworks to memorize up front and no expensive tools to buy. If you want the broader reference material on every methodology, our main guide to [threat modeling](#) covers it in depth. Here, we focus on something narrower: how to do threat modeling, one concrete step at a time, using a small worked example you can copy for your own project.



What You Are Actually Trying to Do

Before the steps, it helps to be clear about the goal. You are not trying to list every possible attack in the world. You are trying to answer four practical questions about one specific system

1. What are we building?
2. What could go wrong?
3. What are we going to do about it?
4. Did we do a good enough job?

Everything in this article is a way of answering those four questions in a repeatable order. The output is not a report to file away. It is a short list of decisions we will add rate limiting here, we will move this secret out of the client, we accept this risk for now because the impact is low.

The Four Building Blocks You Need to Know

You need only a handful of terms to start. Learn these and the rest of the vocabulary will make sense later.

Assets are the things worth protecting customer records, credentials, payment details, uptime, or your reputation. If nothing valuable is at stake, there is nothing to model.

Threat agents are the people or systems that might attack an anonymous internet user, a logged in customer poking at other customers data, a malicious insider, or an automated bot. Naming them keeps the conversation realistic. A curious customer and a well funded criminal group have different capabilities.

Trust boundaries are the lines where data moves between areas with different levels of trust, such as from a browser to your server, or from your server to a third party payment service. Most serious problems live where data crosses one of these lines.

Attack surface is the sum of all the places an attacker can interact with your system: every form field, API endpoint, file upload, admin panel and integration. Shrinking it is one of the most effective defensive moves available and mapping it is part of every model.

Step 1 Draw the System as a Data Flow Diagram

A data flow diagram, usually shortened to DFD, is the single most useful artifact in the whole process. You do not need special software. A whiteboard or a piece of paper works well.

Use only four shapes

- External entities (rectangles) people or systems outside your control, such as a user's browser or a payment provider.

- Processes (circles) pieces of code that do something, such as a web server or a background worker.
- Data stores (two parallel lines) places where data rests, such as a database or object storage.
- Data flows (arrows) the movement of data between the other three.

A worked example a small notes application

To make this concrete, imagine a small web application called NoteShare. Users register, log in, write notes, attach files and share a note with others through a link. Passwords can be reset by email. Here is the system in plain words

- The user's browser talks to a web server over HTTPS.
- The web server reads and writes to a database holding accounts and notes.
- Uploaded attachments go to file storage.
- A mail service sends password reset emails.

Draw four boxes and connect them with labeled arrows login credentials, note content, attachment upload and reset email request. That is the entire diagram. Notice how little it takes.

Step 2 Mark the Trust Boundaries

Now draw dashed lines wherever data crosses from one trust level to another. In NoteShare, you would draw at least three

1. Between the browser and the web server. Everything arriving from the browser is untrusted input, no matter how friendly the interface looks.
2. Between the web server and the third party mail service, because you depend on an outside party.
3. Between the web server and the file storage, if storage is a separate service with its own permissions.

If you are unsure whether a line is a boundary, ask "If the thing on the other side were compromised or malicious, would that change what I should accept from it?" If yes, it is a boundary.

Marking boundaries this way also helps you see your attack surface. Every arrow that crosses a dashed line is a doorway. NoteShare has doorways for registration, login, note creation, sharing links, file upload and password reset. That is six places to examine closely, which is far more manageable than "the whole app."

Step 3 Ask What Could Go Wrong

This is the creative heart of the exercise. To keep it from becoming a random brainstorm, most beginners use a checklist of threat categories. The most popular is STRIDE, developed at Microsoft, which gives you six prompts to apply to each element

Category	The question to ask	Example in NoteShare
Spoofing	Can someone pretend to be someone else?	Guessing or stealing a session token to act as another user
Tampering	Can data be changed without permission?	Editing the note ID in a request to overwrite another user's note
Repudiation	Can someone deny an action with no proof?	An admin deletes notes and no log records who did it
Information disclosure	Can data leak to the wrong person?	A shared link that never expires and can be guessed
Denial of service	Can the service be knocked over?	Uploading enormous files until storage or memory runs out
Elevation of privilege	Can someone gain rights they should not have?	A normal user reaching an admin endpoint

Walk through each doorway from Step 2 and apply the six questions. Do not worry about being exhaustive. Twenty honest threats are more useful than two hundred vague ones.

Look at real vulnerability classes

It also helps to connect your ideas to real, well documented flaws. If your notes support rich text, you should be thinking about [crosssite scripting](#), where a malicious note runs script in another user's browser. If your search feature builds database queries from user input, injection belongs on the list. The public OWASP guidelines are a good source of prompts when your team runs out of ideas.

Step 4 Rank the Threats

You will now have a list that is longer than you can fix immediately. Ranking turns it into a plan.

The simplest workable method is to rate each threat on two dimensions

- Likelihood: How easy is this to do and how many people could do it? Low, medium, or high.

- Impact If it happens, how bad is it for users and the business? Low, medium, or high.

Threats that are high on both go to the top. This is essentially a small risk assessment and it does not need a scoring formula to be useful. Some teams later adopt numeric models such as DREAD or CVSS style scoring, but a three level rating is entirely sufficient for a first model and avoids false precision.

Here is how NoteShare's top items might rank

Threat	Likelihood	Impact	Priority
Changing a note ID to read or edit another user's note	High	High	1
Password reset link that can be reused or guessed	Medium	High	2
Stored script in a note executing in another user's browser	Medium	High	3
No audit log of admin deletions	Medium	Medium	4
Unlimited upload size exhausting storage	Medium	Medium	5

Notice that the most dangerous item is a plain access control mistake, not an exotic technique. That is typical. Ordinary design gaps outnumber clever attacks in almost every real system.

Step 5 Decide What to Do About Each Threat



For every serious threat, you have four honest options

- Mitigate adds a control that reduces the likelihood or impact.
- Eliminate the feature or data that creates the risk.
- Transfer shifts the risk to another party, for example by using a payment provider so you never store card numbers.
- Accept consciously deciding the risk is tolerable and write down who decided.

For NoteShare, the top mitigations might be

1. Objectlevel authorization checks on every note request, verifying that the loggedin user owns or has been granted the note, not just that they are logged in.
2. Singleuse, shortlived, unpredictable reset tokens that are invalidated after use.
3. Output encoding and a restrictive content security policy for note content.
4. Append only audit logging for administrative actions.
5. Upload size limits and quotas per user.

These are security controls and each one maps directly back to a threat on your list. That traceability is what separates a threat model from a generic security checklist. Work through the [handson challenges](#) to test whether the threats you imagine are the ones that actually work.

Step 6 Check Your Work

The fourth question, Did we do a good job?, is the one most beginners skip. Validate the model in three quick ways

1. Coverage check Did every arrow that crosses a trust boundary get examined? If you skipped one, go back.
2. Reality check Does the diagram match how the system actually works today, not how it was designed on paper? Ask the developer who wrote it.
3. Testing check Can each mitigation be verified? Where possible, turn threats into test cases and try them against a running build.

The third check is where practice pays off. Reading that an IDswapping attack is possible is very different from performing it. Running through a purpose built vulnerable target, such as the one in our [OWASP vulnerable web app tutorial](#), helps you internalize how these flaws behave in practice.

Common Beginner Mistakes and How to Avoid Them

Modeling the implementation instead of the design. If you start from code, you tend to find only what the code already does. Start from the diagram and ask what it should be doing, then compare.

Only listing threats you already know how to fix. People instinctively gravitate toward comfortable problems. Push the group to write down uncomfortable ones as well.

Treating it as a one time event. A model that is never updated becomes fiction. Attach a short review to any change that adds a new integration, data type, or user role.

Leaving developers out. The people who build the system know where the shortcuts are. A model produced by outsiders alone is usually incomplete and often ignored.

How Threat Modeling Fits Different Kinds of Software

The steps above stay the same whatever you build, but the details change.

For a web application, focus on authentication, sessions and every place user input enters. For a mobile app, add the device itself as an untrusted environment, since attackers can inspect local storage and intercept traffic. Our overview of [mobile app vulnerabilities](#) is a useful companion when you model a phonebased client. For systems built with heavy help from AI assistants, add the code generation process itself to your list of concerns; the risks are explored in our guide to vibe coding security risks.

In each case, you will discover that the same handful of design questions keep returning who is allowed to do this, how do we know it is really them, what happens if this input is hostile and how would we find out afterward?

How to Build the Skill After Your First Model

Your first model will be rough. That is normal and expected. The skill improves with repetition and with exposure to how attacks actually work.

A practical practice plan looks like this

1. Model something you know well. Use a project where you understand the architecture, so you can focus on the method rather than on learning the system.
2. Attack something on purpose. Work through the intentionally vulnerable applications in our web application hacking labs. Seeing an exploit succeed teaches you what a threat really looks like.
3. Learn one flaw class deeply. The [SQL injection lab](#) is a good candidate because it shows how a small trust assumption in a query can expose an entire database.
4. Read code with a threat in mind. Pick a suspected weakness from your model and search the implementation for it. The source code review labs build exactly this habit.
5. Learn from common failures. Reading about the common application security challenges developers hit in real projects helps you predict where your own designs will be weakest.

Conclusion

Learning how to do threat modeling does not require a certification, a large budget, or a background in advanced security research. It requires a clear diagram, an honest look at your trust boundaries and the discipline to ask what could go wrong here? in a structured way. Draw the system, mark where trust changes, apply a simple checklist such as STRIDE, rank what you find, choose a response for each serious threat and then verify the result.

Start small, involve the people who build the system and treat the model as a living document. Every model you complete makes the next one faster and sharper. When you are ready to turn theory into skill, explore the practical exercises at [AppSecMaster](#), where you can practice attacking and defending realistic applications and see exactly how the threats in your models behave in the real world.

Frequently Asked Questions (FAQs)

How long does a first threat model take?

A focused model of one feature typically takes one to two hours with a small group. Larger systems take longer, but you should still break them into featuresized pieces. If a session runs beyond two hours, reduce the scope rather than extending the meeting.

Do I need special tools to perform threat modeling?

No. A whiteboard, a shared drawing tool, or plain paper is enough to start. Dedicated tools become useful later, when you need to keep many models consistent, store them alongside code, or generate reports for auditors. Begin with the method, then choose tooling to support it.

What is the best framework for beginners?

STRIDE is the most common starting point because it gives you six clear prompts and maps neatly onto a data flow diagram. Once you are comfortable, you can explore attack trees, scoring models, or riskcentric approaches, but you do not need them to produce a valuable first model.

How often should a threat model be updated?

Update it whenever the design changes in a meaningful way: a new integration, a new type of sensitive data, a new user role, or a change to how components communicate. Many teams also schedule a light review every few months so that gradual drift does not go unnoticed.

Is threat modeling the same as penetration testing?

No. Threat modeling examines a design to find weaknesses before or during development. Penetration testing attacks a running system to find weaknesses that exist in the implementation. Each finds problems the other misses, so mature teams use both, with the threat model helping testers decide where to focus.