THE WORKSHOP

# Threat Modelling

A pattern for walking a system diagram with a deliberately hostile eye, using STRIDE as a checklist, so that the ways a system can be abused become visible before the people who'd like to abuse it find them.

2027-01-15
barkingiguana.com/writing/the-workshop-threat-modelling/

A facilitator's playbook — print, annotate, and keep on the table.

# Contents

*The LLM*

*LLMA neural network trained to predict the next token in a sequence, large enough that it generalises to tasks it wasn't explicitly trained for. writes code that passes tests but doesn't think about what could go wrong. STRIDE-style threat modelling walks through every boundary in your system, listing the threats and the mitigations before the code ships. Worked example: What the LLM Didn't Think About.*

## Threat Modelling

Threat modelling walks a system's architecture with a deliberately hostile eye, using STRIDE as a systematic prompt, so that the ways the system could be abused become visible and rateable before they become incidents. Most commonly known as STRIDE threat modelling, after the mnemonic Microsoft popularised in the 1990s (Spoofing, Tampering, Repudiation, Information disclosure, Denial of service, Elevation of privilege); the session hangs on Adam Shostack's four questions (*what are we working on, what can go wrong, what are we going to do about it, did we do a good job*) which the phases below answer in order. Other frameworks exist (PASTA, LINDDUN, attack trees, kill chains), each with its own strengths, but STRIDE is the Toyota Corolla of threat modelling: well-understood, widely taught, reliably effective, and almost always the correct place to start. Occasionally confused with penetration testing (which attacks a running system from the outside) and with security code review (which inspects code for known vulnerabilities); threat modelling happens on the whiteboard, looks at the design, and asks *"what could go wrong,"* not *"what's wrong."*

**At a glance**

- *Who, for how long:* a facilitator who doesn't identify threats, two or more developers who know the system, an operations or SRE person, and ideally a product person plus a security specialist if you have one. Four to six people, two hours.
- *What you walk out with:* an annotated diagram with trust boundaries marked, 20–40 specific rateable threats prioritised by severity, mitigation seeds for the high and critical ones, and a corrected diagram that now matches reality.
- *When to reach for it:* designing a new system or major feature, adding an integration, handling sensitive data, or changing a trust boundary in the infrastructure. Not for systems where nothing material has changed since the last model (review the existing one), where there are no external interfaces and no sensitive data, or where nobody in the room knows how the system actually works (do a technical walk-through first).

## What's It For

A team ships a new API. It works. It's documented. It has tests. Six weeks later, a security researcher emails to say that by changing a single parameter in the URL, any authenticated user can read any other user's data. The team is stunned: how could nobody have noticed? They noticed *authentication*. They didn't notice *authorisation*, which is a different question. Authentication answers *who are you*; authorisation answers *what are you allowed to do*. The difference is exactly the kind of thing that a five-minute STRIDE walk-through would have surfaced before the code was written.

Or: an SRE team adds a new managed service to the stack. The service has its own IAM model, its own network surface, its own logging. The team treats it like a black box because the vendor's documentation says it's secure. Two years later an incident reveals that the service's default configuration exposes an admin endpoint to the internet. The vendor *did* document this, in a page nobody read, because nobody had a habit of asking *"how could this be abused"* about every new piece of infrastructure.

Or: a team is told they need "to do security" before shipping. They run a checklist. The checklist has items like "passwords are hashed" and "HTTPS is enabled." They tick the items. Every item on the list is genuinely important, and the list has nothing to do with the actual attacks their system is vulnerable to. They pass the checklist and ship the holes.

The common thread is that security problems are not found by people thinking about features. They're found by people thinking about abuse, systematically, with a prompt that forces them to consider categories of attack they don't naturally think about. STRIDE is that prompt. The session is the forcing function that makes the prompt get applied. Without the session, the prompt gets applied on Fridays when there's time, which is to say, never.

Reach for it when:

- You're designing a new system or a major new feature
- You're adding a new integration: an API, a webhook, a third-party service, a new data source
- The system handles sensitive data: personal information, payments, health data, credentials, anything the loss of which would trigger a notification letter
- You haven't done one before and the system is already in production (better late than never, and the find rate on a first session is always scary)
- An SRE team is deploying a new service or changing a trust boundary (lines on the diagram where the level of trust changes, e.g. user→server, server→database) in the infrastructure
- You're about to expose something to a new class of user: going from internal to external, from authenticated to public, from one tenant to multi-tenant

## What It's Not For

Skip it when:

- Nothing material has changed since the last threat model. Review the existing one instead of starting fresh.
- The system has no external interfaces and no sensitive data. Rare in practice; be honest about this.
- Nobody in the room knows how the system actually works. Do a technical walk-through first, then threat model.

Stop a session that's already started if:

- An hour in and no threats have been captured; the room can't hold the hostile frame
- Every threat is being downgraded to "we already handle that" without evidence
- The team has shifted from attacking the system to litigating past decisions

Trade-offs to weigh up. The benefits are concrete: a list of specific, rateable threats with the top ones already matched to mitigation ideas; a team that now thinks about attack surfaces, trust boundaries, and data flows as first-class design questions; the habit of asking *"what could go wrong here"* during design, not after; surprises in production that would have taken six weeks to find, found in two hours instead; and a diagram that actually matches reality, because the threat model forced it to be checked.

The costs are real: two hours of four to six people, including people whose calendars are tight; emotional cost when the team discovers a significant issue in code they're proud of; political cost when a finding exposes a decision someone senior made and defended; and the mitigations go into a backlog that already has other things in it. Competing priorities are real.

The common failure modes: the team defends the system instead of attacking it, and the threat list is empty; only dramatic categories get applied, and information disclosure and repudiation go unexamined; mitigations are identified but never executed, turning the session into a documented list of known vulnerabilities with no owner; the diagram is wrong and nobody corrects it, so the threats attach to fictional components.

The deepest benefit of threat modelling isn't the list of threats; it's the habit the team develops of asking *"how could this be abused"* during every design conversation, without being prompted. The formal session is the training ground for that habit. The teams that ship fewer security incidents aren't the ones with the biggest security budgets; they're the ones where every developer, every time, asks the question before writing the code.

## Definitions & Background

STRIDE is the mnemonic at the heart of the session: Spoofing (pretending to be someone else), Tampering (modifying data you shouldn't), Repudiation (denying you did something the system can't prove), Information disclosure (exposing data you shouldn't), Denial of service (making the system unavailable), Elevation of privilege (gaining capabilities you shouldn't have).

Data flow diagrams (DFDs) come with four element types: process (a running thing that does work), data store (a place data sits), data flow (data moving between elements), external entity (something outside the system you're modelling, like a user or a third-party service).

STRIDE per element type. Each element on the architecture diagram has a *type* (process, data store, data flow, external entity) and not every STRIDE category applies to every type. Data stores can't spoof. Data flows can't repudiate. The mapping the STRIDE-per-element discipline uses:

| Element type | S | T | R | I | D | E |
|---|---|---|---|---|---|---|
| Process | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Data store | | ✓ | ✓ | ✓ | ✓ | |
| Data flow | | ✓ | | ✓ | ✓ | |
| External entity | ✓ | | ✓ | | | |

Apply STRIDE to each element only against its matching categories. Without typing, the team applies all six categories to everything and burns out by element five.

Trust boundaries. Lines on the diagram where the level of trust changes: user→server, server→database, internal-network→public-internet, low-privilege-service→high-privilege-service. Every place where data crosses a trust boundary is an attack surface in waiting. Mark them on the diagram before you start identifying threats; they will be the loci of most of the findings.

## Inputs

- A system diagram on the wall. It doesn't need to be a formal architecture document; a whiteboard-level sketch showing components, data flows, and trust boundaries is enough. If no diagram exists, spend the first 30 minutes drawing one together; the absence of a diagram is itself a finding.
- An explicit scope written at the top of the diagram. *"Today we're threat-modelling the subscription pause/resume flow. Everything that touches that flow is in scope. Everything else is parked for another session."*
- Sticky notes for threats; different-coloured sticky notes for mitigations; dot stickers or markers for severity ratings.
- A STRIDE reference visible in the room (a printed sheet, a whiteboard list, a slide) so participants can glance at the six categories without having to remember them.
- A two-hour slot with a real break in the middle, and the right people in the room (see *Who's Needed*).

Bring nothing more special. The diagram, the STRIDE categories, and a hostile imagination are the tools.

## Outputs

What lands on the wall at the end:

- An annotated diagram with trust boundaries marked, threats attached to components as sticky notes, and mitigations attached to the high and critical threats as different-coloured sticky notes.
- A list of specific, rateable threats (between 10 and 40 depending on the level, see below) each one phrased concretely. *"An unauthenticated user can access /subscriptions/{id} and enumerate subscriber IDs"* is a threat. *"API is insecure"* is a complaint.
- Severity ratings for every threat, on likelihood × impact, marked on the sticky with coloured dots or a quick handwritten rating.
- Mitigation ideas for the high and critical threats, each one a seed for a backlog ticket rather than the whole ticket.
- A scope decision for any threats found outside the agreed scope: parked for the next session, not folded in.
- A corrected diagram. The act of threat modelling forces the diagram to match reality, and the corrections are themselves valuable.

Photograph the annotated diagram and all threat/mitigation notes. Multiple angles, good lighting. A blurry photo of a good session is a real loss.

These outputs feed straight into:

- Event Storming. Event Storming's actors-and-commands view is a direct input to threat modelling. Every command becomes a question: *"who triggers this, and what happens if they're not who they claim to be?"* Run Event Storming first when the system is unfamiliar, then threat model the flows it surfaces.
- Decision Tables. When a security decision depends on combinations of conditions (roles, permissions, resource states), a Decision Table makes the combinations explicit and ensures the threat model isn't missing a combination nobody thought of. Pair them when the access-control logic is non-trivial.
- Ensemble Programming. An ensemble session with the threat model on the wall next to the screen is one of the best ways to implement security-sensitive code. Every line is written in the presence of the threats it has to defend against.
- Retrospectives. After a security incident, a post-incident threat-model-style retrospective surfaces the category of the miss as well as the specific miss. *"Which STRIDE category would have caught this, if we'd applied it at design time?"* is a question that changes how the team designs next time.

## Who's Needed

Four to six people, two hours:

- Facilitator. Does not identify threats directly. Their job is to keep the STRIDE categories applied systematically, enforce the hostile framing, prevent the session from turning into a defence of past decisions, and stop the group from skipping categories they find boring.
- Developers who know the system architecture. At least two. They know how data moves, which components exist, and where the bodies are buried. Without them, the threat model floats free of the actual code.
- Operations and SRE people. Essential, not optional. They know how the system is deployed, which network boundaries actually exist versus which ones are on the diagram, and what secrets are sitting in which environment variables. When the session is about an infrastructure change rather than a feature, they lead it.
- A security specialist, if you have one. Valuable but not mandatory. STRIDE compensates for the absence of deep security expertise by making the framework do the prompting. A security person in the room speeds things up and catches things a non-specialist misses; their absence doesn't block the session.
- A product person. Useful for knowing what data matters and what the real-world impact of a breach would be. A threat that leaks subscriber addresses is different in severity depending on whether those addresses are public knowledge or someone's place of refuge.

Below four and you don't have enough angles on attack; above six and the detail work collapses into superficial conversation.

Who to leave out:

- People who take security findings personally. If the developer who wrote the code is going to defend every choice as if it's a personal attack, brief them first or run the session without them. A threat model that's actually a trial produces no threats.
- Senior leaders who'll shut down inconvenient findings. A threat that can't be acknowledged can't be mitigated. If the CTO can't hear "your pet integration has an IDOR bug" (IDOR is Insecure Direct Object Reference, where swapping an ID in a URL or form returns someone else's data) without derailing the session, run it without them and walk them through the findings afterwards.
- Spectators. Threat modelling is participatory, detailed work. Passive observers absorb airtime and find nothing.

## How To Run It

| Phase | Duration | Materials | Key question |
|---|---|---|---|
| Scope framing, diagram review | 15 min | System diagram on a wall | "Is this what we're modelling? Is the diagram accurate?" |
| Identify threats with STRIDE | 45 min | Sticky notes, STRIDE reference | "What could go wrong at this component?" |
| Break | 10 min | , | , |
| Rate severity | 20 min | Dot stickers or markers | "How likely? How bad?" |
| Identify mitigations | 15 min | Different-coloured sticky notes | "What's the simplest fix for the worst threats?" |
| Wrap-up, owners, next steps | 15 min | , | "What goes in the backlog this week?" |
| Total | 2 hours | | |

The active work is 95 minutes, the break is 10, and the rest is scope-setting and wrap-up. Phase 2 is the hard phase (the hostile framing is cognitively expensive) so the break after it is non-negotiable.

Choose the level before the session. The level changes the scope and the energy required.

| Level | Scope | Duration | Output |
|---|---|---|---|
| Feature Level *(zoom in)* | One new feature or one new integration | 60-90 min | 10-20 threats, mitigations for top 3-5 |
| Standard *(default)* | One subsystem, data flow, or trust boundary | 2 hours | 20-40 threats, prioritised list, mitigations for top items |
| System Level *(zoom out)* | An entire system end to end | Half day, often split | Attack surface catalogue, structural risks, architectural changes identified |

The Standard level is almost always the correct starting point. A specific boundary (*everything that crosses the public API,* or *everything that touches subscriber payment data*) narrow enough to be thorough, broad enough to find real issues. Feature Level is worth running every time a new feature lands in the review queue. System Level is a once-a-year exercise, usually after a reorg or a major architectural change.

### Defend, then attack

The session has one awkward but necessary shift: the team spends the first phase defending the system (is the diagram correct, does everyone agree it's accurate) and then switches to attacking it. The shift is not natural. The facilitator's job is to make the switch explicit and to hold the attacking frame hard.

The rhythm is walk the diagram, apply STRIDE to one component, capture what you find, move to the next component. Systematic. Methodical. Boring in the best way. Surprises come from the systematic coverage, not from anyone's brilliance.

Developers will find threats in components they know well. SRE will find threats at boundaries and in configurations. The product person will find threats whose impact the technologists are under-weighting. The security specialist, if present, will find the categories the others are skipping. The roles don't need to be enforced; let the session flow and prompt anyone who's gone quiet.

### Phase 1. Scope framing, diagram review (15 min)

Put the system diagram on the wall before the session starts. Write the scope at the top of the diagram, explicitly:

> *"Today we're threat-modelling the subscription pause/resume flow. Everything that touches that flow is in scope. Everything else is parked for another session."*

Then walk the diagram with the group. For each component and data flow, confirm:

- What data moves through it?
- Who can access it?
- What authentication and authorisation exist?
- Where are the trust boundaries?

Mark the trust boundaries on the diagram with a different colour. Every place where data crosses a trust boundary is an attack surface in waiting.

What to say:

> *"For the next fifteen minutes, the job is to check the diagram is correct. After that, we're going to switch modes. We stop defending the system and we start attacking it. I'll call the switch."*

> *"Where are the trust boundaries? Anywhere data crosses from a less-trusted zone to a more-trusted one, I want a line drawn."*

> *"If the diagram doesn't match reality, fix it now. The diagram is a tool, not a sacred document."*

What to watch for:

- The diagram is wrong. Normal. Developers will correct it in real time. Update the diagram; what's on the wall has to match what's in production.
- Missing components. *"What about the admin panel?" "Oh right, I forgot about that."* Components forgotten in diagrams are forgotten in threat models, and then they're forgotten in security generally. Capture them.
- No trust boundaries at all. If nobody can point at where trust changes, that's a significant finding before you've even started. Everything can't be equally trusted; if it is, that's the first thing to mitigate.
- Assumed security. *"The API is secure because it uses HTTPS."* HTTPS protects the transport; it says nothing about authentication or authorisation. Push for specifics: *"What does HTTPS protect against, exactly? What doesn't it cover?"*

### Phase 2. Identify threats with STRIDE (45 min)

Announce the shift:

> *"Okay. From now until the break, we're attackers. We are not defending this system. Every time someone says 'well, we don't really need to worry about that,' I'm going to push back. Our job is to find the weak points. Every system has them. The question is whether we find them or someone else does."*

Then walk the diagram systematically. For each component and data flow, apply the six STRIDE categories out loud. Pick a component, say, the subscription management API, and ask:

Spoofing. *"Can someone pretend to be a different subscriber? Can a request pretend to come from our mobile app when it doesn't? Can an attacker impersonate a service account or a scheduled job?"*

Tampering. *"Can someone modify a subscription request in transit? Can someone change data in the database directly? Can someone tamper with a build artifact or a deployment?"*

Repudiation. *"If a subscriber claims they never paused their box, can we prove otherwise? Do we have audit logs? Are the logs tamper-evident? Can an admin action be traced to the specific person who performed it?"*

Information disclosure. *"Can someone see another subscriber's address? Can someone enumerate all subscribers through the API? Are error messages revealing database structure? Are we logging credentials?"*

Denial of service. *"Can someone make the API unavailable by flooding it with requests? What happens if the database runs out of connections? Can a single subscriber consume all the capacity?"*

Elevation of privilege. *"Can a regular subscriber access admin functions? Can a read-only API key be used to write data? Can a low-privilege service account reach something it shouldn't?"*

For each threat identified, write it on a sticky note and place it next to the relevant component on the diagram. Be specific. *"An unauthenticated user can access /subscriptions/{id} and enumerate subscriber IDs"* is a threat. *"API is insecure"* is a complaint.

Move systematically through the diagram. Don't try to be exhaustive on every STRIDE category for every component; spend the time where the risk is highest (at trust boundaries, around sensitive data, and at user-facing interfaces). But *do* apply every category at least once to every important component. Teams gravitate toward spoofing and elevation of privilege because they're dramatic; repudiation and information disclosure are less exciting and often more important.

What to say:

> *"Apply all six categories to this component, one by one. I know it feels repetitive. The repetition is the point; if we don't apply every category, we'll find the ones we always find and miss the ones we always miss."*

> *"Be specific. Not 'auth is weak.' 'An attacker with a stolen session cookie can do X, Y, Z.'"*

> *"Assume the network is secure. What can an authenticated user do that they shouldn't?"*

> *"If a subscriber calls support and says 'I never cancelled,' can we prove they did? Walk me through how."*

What to watch for:

- "We're fine because we use [framework]." Maybe. Push: *"What specifically does it protect against? What doesn't it cover?"* Frameworks protect against specific, documented things. Knowing the list is part of the threat model.
- Threat fatigue. After thirty minutes people run out of ideas. Change context: switch to a different part of the diagram, or switch STRIDE category. The pattern break refreshes thinking.
- Network-only threats. The team is finding man-in-the-middle and DDoS attacks but ignoring application-level issues. Redirect: *"Assume TLS is working. What can a legitimate, authenticated user do that they shouldn't be allowed to?"*
- Skipping repudiation and information disclosure. These are the categories most often under-weighted. Make sure each component gets at least one pass through both.
- "Nobody would bother attacking us." Attackers use automated tools. They don't care about company size. Judge threats by impact and exploitability, not by whether you feel important enough to be targeted.
- The apologetic finding. A developer says *"I mean, I guess someone could…"* and then trails off. That's exactly the kind of threat worth capturing. *"Write it. We rate it later. Everything gets captured first."*

### Phase 3. Break (10 min)

Take a real break. The hostile framing is draining, and the severity phase needs fresh minds. Walk away from the wall.

### Phase 4. Rate severity (20 min)

Go back to the wall. For each threat sticky, rate it on two dimensions.

Likelihood.

- Low: requires insider access, advanced skills, or a precise combination of prior conditions
- Medium: exploitable by a moderately skilled attacker, or there's a known attack pattern
- High: easily exploitable, automated tools exist, no skill required

Impact.

- Low: minor inconvenience, no data loss, no regulatory or reputational consequence
- Medium: partial data exposure, temporary service disruption, some recoverable damage
- High: full data breach, financial loss, regulatory consequences, significant reputational damage

Combine them into a simple severity grid:

| | Low impact | Medium impact | High impact |
|---|---|---|---|
| High likelihood | Medium | High | Critical |
| Medium likelihood | Low | Medium | High |
| Low likelihood | Low | Low | Medium |

Mark severity on each sticky note with coloured dots or a quick handwritten rating. Move fast: quick show of hands for likelihood, quick show of hands for impact, read the grid, mark the note, move on. Don't debate each threat for five minutes. If the group is genuinely split, mark the note with a question mark and move on.

What to say:

> *"Likelihood: low, medium, or high? Show of hands. Impact: low, medium, or high? Show of hands. That's your rating. Move on."*

> *"If everything comes out critical, the rating is useless. Force-rank: if we could only fix one, which one? That's critical. The rest are below it."*

> *"Personal data is sensitive. Full stop. Don't downplay a subscriber-address leak by saying 'it's just addresses.' How would our subscribers feel if their addresses appeared on a public spreadsheet?"*

What to watch for:

- Everything is critical. The rating has collapsed. Force-rank: *"If you could only fix one, which one?"*
- Everything is low. The team is protecting the system's reputation by downplaying findings. Push: *"If an attacker actually did this, what would the headline say?"*
- The five-minute debate. Move on. A rating that's close to wrong is still useful; a session that runs out of time never produces mitigations.

### Phase 5. Identify mitigations (15 min)

For the high and critical threats, brainstorm mitigations. Write each mitigation on a different-coloured sticky note and place it next to the threat it addresses.

Mitigations come in three kinds:

- Technical. Rate limiting, input validation, encryption at rest, access control changes, scoped tokens, network segmentation.
- Process. Security reviews, penetration testing, incident response planning, approval flows for sensitive operations.
- Monitoring. Audit logging, anomaly detection, alerting on unusual patterns, honeypots.

Don't design the full fix in the session. Capture the *kind* of mitigation and enough detail for someone to act on it next week. The mitigation sticky is the seed of a backlog ticket, not the whole ticket.

What to say:

> *"What's the simplest fix for this threat? Not the perfect fix, just the simplest one that reduces the severity to something we can accept."*

> *"Match the mitigation to the category. Encryption doesn't prevent elevation of privilege. Rate limiting doesn't stop information disclosure. Each category has its own class of fixes."*

> *"If the mitigation for a medium-severity threat requires a month of work, it's not proportionate. Find a smaller one."*

What to watch for:

- "We should encrypt everything." Encryption is a mitigation for information disclosure and tampering. It does nothing for denial of service or elevation of privilege. Match the medicine to the disease.
- Mitigations bigger than the feature. A one-month fix for a medium-severity threat usually isn't the correct answer. Look for a cheaper partial mitigation that reduces severity, plus a watch-list entry for the fuller fix.
- Accepted risk without documentation. Some threats will be accepted: too low-severity to fix now, too expensive to mitigate. Fine. But write it down: *"We accept this risk because…"*. Undocumented acceptance becomes forgotten acceptance, which becomes a 3am surprise.

### A worked example

See Threat Modelling: What the LLM Didn't Think About: the Greenbox team running STRIDE against their new LLM integration and finding the category of attack the model had cheerfully failed to consider. The moment the team realises the LLM had been asked to help design the feature but had never been asked to attack it is the moment the pattern earns its cost.

## What Can Go Wrong

The security expert dominates. The one person in the room with a security background is doing all the talking, and the developers have gone quiet. *Recovery:* Redistribute. *"Let's hear from the developers. In this code, where do you feel least confident about security? What keeps you up at night?"* The security person can follow up; they shouldn't lead. *Stop if:* The developers stay silent even when prompted. The psychological safety isn't there, probably because the security person is perceived as judging. Reframe explicitly or reschedule without them.

The team gets defensive. Every threat is met with *"yes, but we already handle that because…"* Nothing gets captured because everything is already fine. *Recovery:* Reframe hard. *"We're not looking for who made mistakes. Every system has weak points. We want to find ours before someone else does. A finding is a win, not a criticism."* *Stop if:* The defensiveness continues. The team can't hold the hostile frame in a room with the person who built the system. Run the session without that person and walk them through the output afterwards.

Scope creep. The group starts threat-modelling components outside the agreed scope because something juicy is visible next to it. *Recovery: "Great catch. Pink-note it for the next session. Today we're on [scope]."* *Stop if:* The scope was wrong and the interesting threats are all outside it. End the session early, rescope, and reconvene.

The tool debate. Someone suggests using a specific threat-modelling tool or framework instead of pen-and-paper-and-STRIDE. *Recovery: "For a first session, the value is in the conversation, not the artifact. We can revisit tools after we've done this a few times."* Pen and paper beats any tool for a team that hasn't done threat modelling before. *Stop if:* The debate continues. Someone is using the tool question to avoid the actual work. Name it.

The "we can't fix this" despair. The team finds a fundamental architectural issue that can't be mitigated without a major rewrite. *Recovery:* Don't panic. Capture the threat as a long-term risk, identify any short-term mitigations that reduce severity (monitoring, rate limiting, tighter scoping), and plan the architectural work separately. *Stop if:* Half the findings are architectural and the team is paralysed. Stop the identification work and switch to an architectural conversation; reconvene for mitigations once the architecture is decided.

The category skipper. The team is enthusiastic about spoofing and elevation of privilege but keeps gliding past repudiation and information disclosure. *Recovery:* Slow down. *"Every component gets at least one pass through every category. Repudiation for this one: if a subscriber says they didn't do X, can we prove they did?"* Enforce the pass. *Stop if:* The team literally cannot answer the repudiation question for any component. That's the finding: you have no audit trail. Stop and capture that as a critical threat on its own.

## Next Steps

The session ends; the work begins.

Same day, the facilitator:

- Photographs the annotated diagram and all threat/mitigation notes. Multiple angles, good lighting. A blurry photo of a good session is a real loss.
- Transcribes the threats into the security backlog or tracker, with severity ratings and the mitigation ideas attached.
- Sends findings to all participants and to whoever owns the security posture for this product.
- Flags any critical findings immediately, without waiting for the formal backlog triage. A critical that exists for another week because the document was pending isn't security, it's paperwork.

This week, the product owner:

- Turns critical and high threats into backlog items, treated as bugs. Near the top of the backlog, not the bottom. *"Fix IDOR on /subscriptions/{id}"* is a bug, not a feature, and it should be prioritised accordingly.
- Triages the medium and low threats. Some will become backlog items; others will become *"accepted risk, documented, reviewed in six months."* Don't leave them in ambiguous limbo. Every threat gets a disposition.

- Matches mitigations to owners. Each mitigation has one person responsible for getting it done. Not a team, not a role: a name. A mitigation without a name ages on a list.
- Schedules the follow-up session for anything out of scope. If the team kept saying *"that's interesting but out of scope,"* book the next session now, while the momentum is fresh.
- Shares findings upward. Senior stakeholders need to know the shape of the risks, not every detail. A one-page summary with the top three findings and the mitigation plan is the correct unit.

Ongoing, the team:

- Re-runs the threat model when the architecture changes significantly: new components, new integrations, new data types, new trust boundaries.
- Builds a fifteen-minute STRIDE walk into the design review for every new feature. It doesn't need to be a formal session; a quick "what could go wrong" pass at the whiteboard is enough. The full session is for larger changes; the mini-version is for everything else.
- Tracks mitigations to completion. Identified threats that never get mitigated are worse than unidentified ones: you knew, and you did nothing. A spreadsheet is fine; an unmanaged issue tracker is fine; a place where nothing gets forgotten is essential.
- Revisits accepted risks on a schedule. Risks accepted in one context may become unacceptable in another. An annual review catches the ones that quietly changed severity.

## Variants

Standard *(default)*. One subsystem, data flow, or trust boundary, two hours, four to six people. Output: 20-40 threats, prioritised list, mitigations for top items. This is what most teams need, and the rest of this post describes it.

Feature Level *(zoom in)*. One new feature or one new integration, 60-90 minutes, three to four people. Output: 10-20 threats, mitigations for the top three to five. Worth running every time a new feature lands in the review queue.

System Level *(zoom out)*. An entire system end to end, half a day (often split across two sittings), the wider team plus security and operations leads. Output: an attack surface catalogue, structural risks, architectural changes identified. A once-a-year exercise, usually after a reorg or a major architectural change.

LINDDUN for privacy. Same shape as STRIDE per element, but the categories are privacy-focused: Linkability, Identifiability, Non-repudiation, Detectability, Disclosure of information, Unawareness, Non-compliance. Reach for it (alongside or instead of STRIDE) when the asset is personal data and the threat surface is privacy harm rather than security compromise. The session structure transfers cleanly; only the prompts at each component change.

The thirty-minute mini-version. A "what could go wrong" pass at the whiteboard during a design review, with one person calling STRIDE categories and the team responding. No formal sticky notes, no severity grid: just the prompt applied out loud. Use it for every new feature, in addition to (not instead of) the full session for larger changes. The mini-version is what builds the habit; the full session is what makes the mini-version reliable.

Remote. A Miro or Mural board with the diagram pinned, sticky-note swimlanes for each STRIDE category, video call for the conversation. Slightly slower than in person (the rhythm of *"point at component, call category, capture sticky"* is faster around a wall) but the structure transfers. Use one shared cursor: only the facilitator places notes, prompted by the team, to keep the layout legible. The break in the middle matters more remotely, not less.

## About this playbook

This playbook is part of *The Workshop*, a reference series of facilitator playbooks published at barkingiguana.com. The canonical, up-to-date version lives at barkingiguana.com/writing/the-workshop-threat-modelling/.

*These posts are LLM-aided. Backbone, original writing, and structure by Craig. Research and editing by Craig + LLM. Proof-reading by Craig.*