

() **Threat Modeling of Threat Modeling #meta, V 1.0.0**

Available online at <https://threat-modeling.net/threat-modeling-of-threat-modeling/>

Created by Hendrik Ewerlin - <https://hendrik.ewerlin.com/security> - 2024-02-24

Motivation / About this document

This document threat models threat modeling. #meta

Threat modeling will more likely be a success if we tame the threats to the threat modeling process.

Why is threat modeling so important?

Threat modeling is crucial for building secure systems:

A system is secure, iff it is protected from danger.

So the obvious questions are: What danger / threats? What protection / mitigations?

These questions align nicely with Questions 2 and 3 from Shostack's Four Question Framework: "What can go wrong? What are we going to do about it?" Answering these four question is what a threat modeling process does.

This makes threats visible (Goal 1: Clarity) and tames them (Goal 2: Security).

What makes threat modeling a success?

The ultimate goal is to create a secure system. We create that system by threat modeling and implementing mitigations. This is a security activity performed by humans and we can investigate it's usability (see [ISO 9241-11:2018](#)).

1. **Effectiveness:** We need the complete thing. We need threat modelers to finish the threat modeling and developers to finish mitigations. If the process gets stuck somewhere, or the mitigations are not implemented, we end up with nice conversations and plans, but zero improvement of the system.
2. **Efficiency:** We want good quality and acceptable effort.
3. **Satisfaction:** We want everybody to enjoy the activity, so they will love to do it again.

Why threat model threat modeling?

So we need effectiveness, efficiency and satisfaction in a process with multiple steps, different ways to do things and various people involved. Obviously, a lot of things can go wrong. What are we going to do about it?! Hey, isn't that threat modeling? 😊

Methodology

This meta threat model chooses a very simple threat modeling style and notation with four to five levels.

- **0-4** Phase of threat modeling, ***** for overarching aspects
 - 🧩 Typical activity in that phase

- 📁 Threat cluster - optional
 - ☁️ Threat when performing the activity
 - 🌂 Mitigation of that threat

There are short names and detail texts. Threats have stable numerical IDs for referencing that follow the hierarchical structure. [X.Y.Z]

Whenever possible, captions use active voice and clarify performing actors as the subject of sentences. Sometimes, "⇒" denotes how causes result in consequences and have impact.

For simplicity, mitigation texts assume that advice can reach it's destination and be followed. In the real world, this means communication, persuasion and finding better alternatives along the way.

Highlight markers - ✨ - show my personal favorites.

Incompleteness markers - ? - show where mitigations have not yet been considered or documented.

▼ 💡 Inspiration

The analysis has 5 phases. Phase 0 is "How do we threat-model?". Phase 1-4 are the questions from Shostack's Four Question Framework.

The activities are taken from threat modeling approaches as seen online.

Threats were inspired from systematic analysis, own experience, online problem reports, the Threat Modeling Manifesto and feedback. Taking part and following the OWASP #threat-modeling Slack Channel and the Threat Modeling Connect Community was especially helpful and inspiring. Other helpful content is linked.

The original document was vendor-specific and then generalized and extended to target a broader audience of "people at system vendors wanting to succeed with their threat modeling program".

▼ 📁 Why add Phase 0 "How do we threat-model"?

Threat modeling won't happen by accident. Vendors must actively choose to practice it. They need a process. Various approaches exist. There must be a selection, adjustment for local needs and design of parts that are missing.

A threat modeling program takes this one step further. More than a process, a program has considerations about training, education and decisions on who threat models what and when.

How this may apply to you

Mitigations are my own suggestions. You may be concerned about other threats or prefer other mitigations. Some aspects are controversial. Some advice won't work depending on local situations. The threat model has no likelihood considerations. You may want to consider for yourself if a threat applies for you and if you want to try a suggested mitigation.

Take this with a grain of salt. **Get inspired. Enjoy!** 😊

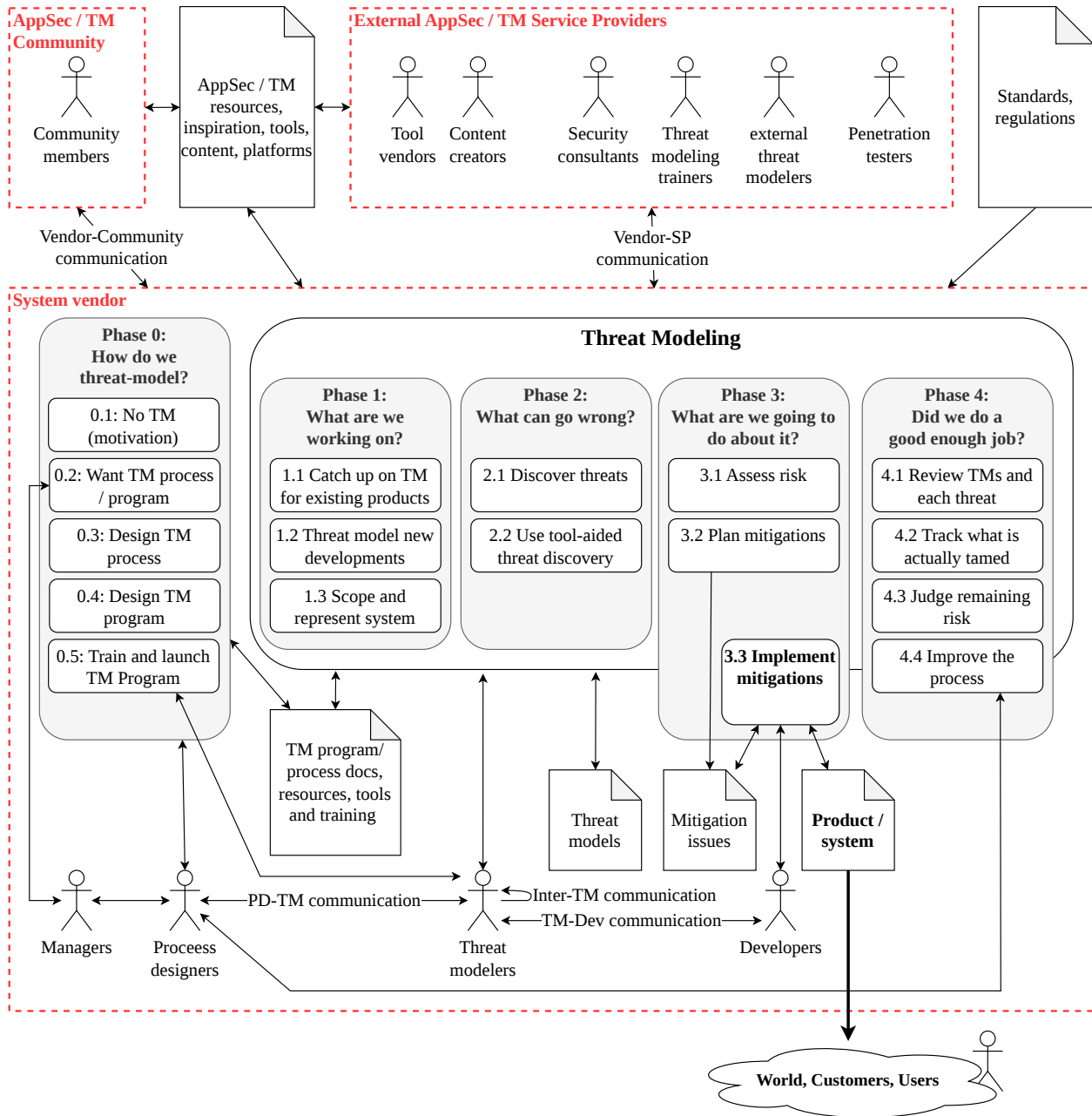
If you have any feedback, please let me know. The bottom of this document has details about the call for feedback.

System Model

- **Actors (with overlaps - a given person may take on one or more roles in this list)**
 - At the system vendor...
 - Managers request or grant threat modeling
 - Process designers define how it is done
 - Threat modelers threat-model
 - Developers implement mitigations

- From remote...
 - AppSec / Threat Modeling community members inspire and support with mutually enriching exchange
 - AppSec / Threat Modeling service providers inspire and support with their services
 - World, Customers, Users benefit from clarity and a secure product
- **Artifacts**
 - At the system vendor...
 - Threat modeling program/process documents, resources, tools and training material empower threat modelers
 - Threat models capture and drive threat modeling processes and results
 - Mitigation issues drive the implementation of mitigations by developers
 - Product / system is shaped, secured and delivered
 - From remote...
 - AppSec / Threat Modeling resources, inspiration, tools, content and platforms inspire, support and may be adopted
 - Standards and regulations apply and inform
- **Phases and activities**
 - See **Diagram**
 - See **Threats and Mitigations** chapter structure

Diagram



Threats and Mitigations

0 “How do we threat-model?” Phase

0 Develop software without threat modeling (May serve as motivation) [0.1]


▼ Is this fear mongering?

The ultimate goal of security ambitions is to make actual damage less likely and less harmful. So, naturally, we also have to talk about insecurity. The focus here is the transformation and goal we are trying to achieve - from blindness to more clarity, from insecurity to more security. We promote the umbrella, not the storm.

▼ Is this over-simplified?

The actual state of vendors and products is highly complex, ambivalent, depending on various aspects. This is also why we need analysis. And why there is room for improvement.

▼ [0.1.1] **Blindness**

- Product was designed without threat modeling. ⇒ There is no way to answer if it is secure (= protected from danger). Gut feeling / known security issues / previous incidents reported / last penetration test report is far inferior than showing a completed and up-to-date threat model with mitigations implemented. Knowledge about insecurity is often distributed and implicit. Depending on who is asked, there may be contradicting assessments of the current state.
-  Threat model and know the security of your system.
(Goal 1 - Clarity - from above)

▼ [0.1.2] **Insecurity**

- Product was designed without threat modeling. ⇒ The product is more likely to be insecure. It is accidentally secure, at best. Developers had the right intuition here and there and implemented proper mitigation of threats. In other cases, this was probably forgotten. This results in systems with “rocket-proof doors” (mitigation overkill, see below) and “open windows” (mitigation underkill, see below).
- 🌂 Threat model, implement mitigations and secure your system. (Goal 2 - Security - from above)

▼ 🌧️ [0.1.3] **Actual Damage**

- The more insecure a product is, the more likely is an actual damage or incident.
- 🌂 Threat model, reduce insecurity and make actual damage less likely and less harmful.

▼ 🌧️ [0.1.4] **Late Threat Modeling**

- Threat modeling is late ⇒ Vendors face the dilemma of choosing between ongoing blindness and insecurity VS threat modeling a “giant” with the consequences associated (see below).
- 🌂 There are two best times to threat model: at design time, and right now.

0 🧩 **Want threat modeling process / program [0.2]**

▼ ? 🌧️ [0.2.1a] **Weak security culture / “I don’t care about security”**

- Managers or developers don’t value security very much. ⇒ It is therefore difficult to convince them of a methodology that promotes it.
- ...?

▼ 🌧️ [0.2.1b] **Unknown or unwanted threat modeling**

- Managers or developers don’t know threat modeling exists / don’t demand or support threat modeling / don’t see that they totally need it

- 🌂 Threat modeling benefits from ambassadors at the vendor who see it's value and promote it.
 - 🌂 Use convincing material provided by the threat modeling community. In this document, for instance, see [Why is threat modeling so important?](#) and [Develop software without threat modeling](#).
 - 🌂 Some experts already highly recommend threat modeling:
 - [OWASP SAMM](#) has [Threat Assessment \(owaspsamm.org\)](#) as a crucial part in the design phase of secure software development and defines three maturity levels.
 - [OWASP Top 10:2021](#) has [A04 Insecure Design - OWASP Top 10:2021](#) on rank 4 of their most important security problems and highlights the importance of threat modeling.
 - 🌂 For some countries and domains, threat modeling is required.
- ▼ ✨🌂 [0.2.2] **"Resistance is futile" misconception**
- Managers and developers sometimes think that defense is futile anyway: According to them, hackers, service providers, governments and future super-computers can easily intrude and steal data. ⇒ Such a mindset results in giving up too early and not properly defending.
 - 🌂 They need security education. State-of-the-art mitigation is extremely powerful and can resist even powerful attacks. It can sometimes achieve levels of security that go far beyond what people intuitively think is possible:
 - Practically unbreakable encryption exists
 - Post-quantum cryptography is currently being standardized
 - Usable secure authentication exists
 - Services with well-crafted end-to-end encryption can provide awesome utility without server intruders or platform hosts being able to access or modify content.
 - Signatures and blockchain technology can make it impossible to tamper with data without being noticed

- Confidential computing enables confidential execution of arbitrary programs in untrusted environments
- Secure build and deployment chains can to some extent resist rogue developers and admins and supply-chain attacks
- 🌂 Embrace cryptography and advanced mitigations and learn what is possible.
- 🌂 Stop building systems with all-mighty admins and “server intruder ⇒ game over” properties.
- 🌂 Hire application security experts.
- ▼ **? 🌂 [0.2.3] “Won’t happen to us” / “We’re invulnerable” misconception**
 - Managers or developers already feel like they are in a good state, nobody would try to attack them and current practices suffice.
 - ...?
- ▼ **🌂 [0.2.4] Undefined desired level of security**
 - The organization lacks specifications or service level agreements (SLAs) how much security is really needed and aimed for. ⇒ As a result, opposing forces are fighting it out on a daily basis. The organization wastes a lot of effort negotiating. People get frustrated. The level of security depends on who can assert themselves and is kind of random. It can depend to a large extent on the composition of the people in teams and who is in charge of prioritizing.
 - Tanya Janca’s DevSecOps Worst Practices talk answers a question about appropriate level of security and also draws a realistic picture about how people are persuading all the time when no standards are set.
 - 🌂 Managers must clearly set expectations how much security is needed. The vendor’s domain, asset considerations, product strategy, current state of security and legal requirements answers questions about what is appropriate.
 - 🌂 If managers don’t set expectations, process designers should promote and demand it.

- 🌂 These expectations must be well communicated and accepted. Even if it will always be the case that security is more or less important to particular people, it must be clear what the organization as a whole is striving for.
- 🌂 If a certain level of security is required, processes must account for that and demand it.
- 🌂 Resource allocation is essential.

0 🧩 Design Threat Modeling process (how) [0.3]

▼ ✨🌂 [0.3.1] No process available

- No actionable process (let alone: program) is available, so no-one threat-models.
- 🌂 Start with some approach from the internet that is actionable and produces some results. Focus on learning and satisfaction first (the experiment helps learn, people enjoy it and get excited), effectiveness second (the process finishes with some result), efficiency third (the quality is good and it doesn't produce too much effort).
- 🌂 Learn what works for you. Learn what doesn't work. Integrate and improve.

▼ 🌂 [0.3.2] Perfect process trap

- Process Designers take forever to design a very sophisticated threat modeling process / program and don't get this whole thing started.
- 🌂 Follow the [Threat Assessment \(owaspsamm.org\)](https://owasp.org/www-project-samm/) maturity levels from [OWASP SAMM](https://owasp.org/www-project-samm/) as a plan for growth. Don't try to jump from maturity level zero to maturity level three.
- 🌂 Before designing a whole threat modeling program, experiment and try single/different threat modeling approaches
- 🌂 Embrace the idea behind the quote "Version one is better than version none", apply the mindset you know from agile software development, start

with a minimal viable process, improve. You may even apply the ideas of [Semantic Versioning 2.0.0](#) to your process.

▼ 🌧️ [0.3.3] **No driver**

- Company lacks threat modeling / application security experts who can be the process designers
- 🌂 Hire application security experts, get consulting or follow out-of-the-box advice from the threat modeling community. Whatever you do, make sure there are drivers who create an actionable threat modeling process.

▼ ✨🌧️ [0.3.4] **Information overkill**

- Process Designers are overwhelmed by various approaches, tools and information that exists out there, while trying to learn Threat Modeling
- 🌂 Refer to hard threat discovery and hard mitigation planning mitigation for some hints to approachable material. (see below)
- 🌂 Start simple.
- 🌂 The Threat Modeling community can support this by creating good material for starters.
- 🌂 The Threat Modeling community can support this by welcoming two types of learners:
 - Some want something simple that is approachable and just works, because they want the results in the first place. Those can use prompts, threat modeling games, cue cards or very simple advice.
 - Some want to dive deep and become expert threat modelers. Those can probably handle huge enumerations and long link lists with rich material.

▼ 🌧️ [0.3.5] **Awesome tool trap**

- Process Designers get lost thinking about tooling
- 🌂 Don't use sophisticated tooling at first.
- 🌂 Simple approach: Start with threat modeling that "hacks the whiteboard" and has mitigation issues as it's outcome.

- 🌂 Use the same tools that are used for other technical documents. Use some diagramming software that is available - draw.io is a good one. Create a template how to denote threats and mitigations.
- 🌂 Test and adopt tooling as you proceed.

▼ ☁️ [0.3.6] **Non-actionable process**

- Process Designers design a process that is not actionable and executable
- 🌂 Have step by step instructions or checklists that guide threat modelers when threat modeling. Don't just describe outcomes, because this results in threat modelers doing process design work at times when they should better be threat modeling.
- 🌂 Reflect on the process (Improve the process with lessons learned, see below)

0 🧩 **Design Threat Modeling program (who / when / what) [0.4]**

🚧 *TODO - this section is under construction and has some ideas. Threat mitigations in the sections "Catch up on threat models for existing products", "Threat model new developments" and other sections reveal several topics that should better be considered by process designers up-front.*

📁 **Who**

▼ ? ☁️ [0.4.1] **Who threat models?**

- Process Designers fail to decide who threat models. ⇒ There's unclear responsibilities and no action.

▼ ? ☁️ [0.4.2] **Few threat modelers**

- Process Designers let too few people threat-model. ⇒ This results in hero threat modeler anti pattern.

▼ ? ☁️ [0.4.3] **Everyone threat modeler**

- Process Designers let everyone threat-model. ⇒ Some people don't want to. Some people simply can't. ⇒ They need a lot of training or deliver bad results.

Catch up plan

▼ [0.4.4] Lack of catch up plan

- Program design lacks a strategy how to catch up on threat modeling for existing products.

Threat modeling / SDLC integration for new developments

▼ [0.4.5] Bad process integration for new developments

- Process designers don't integrate Threat Modeling in existing development life cycles. ⇒ Threat Modeling is not performed due to lack of triggers.
- ...?

▼ [0.4.6] Bad process integration for new products

- Process designers fail to specify how Threat Modeling accompanies the development of new products.
- ...?

Threat Model Confidentiality

▼ [0.4.?] Threat Model Leak

- Process Designers (or threat modelers) fail to restrict access to threat models. ⇒ A threat model in the wrong hands can serve as an attack plan, especially when mitigations are not yet found or implemented.

Train and Launch Threat Modeling program [0.5]

▼ ☁️ [0.5.1] **Lack of training**

- Too little training leaves threat modelers uncertain what to do.
- ☂️ Introduce mandatory training for threat modelers. Process designers, experienced threat modelers or consultants can be the trainers.

▼ ☁️ [0.5.2] **Too theoretical training**

- A very theoretical training fails to convey practical skills.
- ☂️ Include practical threat modeling exercises in the training.
- ☂️ Consider starting with a real world scenario, so that people can relate.
- ☂️ Design a small toy scenario with high impact threats that match your domain, so that people are engaged and see why threat modeling helps.
- ☂️ Follow Chris Romeo's rule: He's not allowed to talk about threat modeling for more than 30 minutes until people have to threat model. 😏 If you break that rule, know why.

▼ ☁️ [0.5.3] **Bad plan or communication**

- Bad communication when to start, how to start and with which activities to start results in no action.
- ☂️ Create a launch plan with deciders. Prioritize work. (Catch up on threat models for existing products, see below) (Threat model new developments, see below) Communicate.

▼ ☁️ [0.5.4] **No obligation or willingness**

- Threat modelers do not threat-model because they are not obliged to do so and are not motivated by their own initiative.
- ☂️ Threat modeling must be made mandatory. Convince managers if needed.
- ☂️ Ignite the passion for threat modeling with good motivation (see above)

1 “What are we working on?” Phase

1 🧩 Catch up on threat models for existing products [1.1]

▼ ✨🌧️ [1.1.1] Threat modeling a “giant”




- A huge product without an existing threat model or with big gaps in threat modeling coverage leaves threat modelers overwhelmed, not knowing where to start. They feel like it will take forever to threat-model the system and implement mitigations. It’s like “trying to boil an ocean”.
- 🌂 Involve a **powerful** set of well-trained **people**.
- 🌂 Have **management support**. If needed, advertise the benefits of threat modeling the “giant” (see above) and the dangers of not catching up (see below).
- 🌂 **Appreciate** what already exists. You are not starting from scratch. Import existing mitigations into your analysis and see where they help.
- 🌂 Use **encouraging language** that supports the attitude that cutting and threat modeling the “giant” is valuable and feasible, step by step. “Giant” itself is already a word that supports the impression of overload. Even worse terms exist out there, like “big ball of mud”. Listen consciously and think about what words convey. Get people excited about a brighter future.
- 🌂 Address **hearts and minds**. In your communication, be aware that this topic has a strong emotional component to it: It is very much about fear, overload, uncertainty, shrinking back, not knowing where to start and escaping into “One fine day → big rewrite / new product → awesome security” fantasies.

- 🌂 **Divide and conquer:** Decompose and cut the “giant” into manageable pieces.
 - 🌂 Choose a **level of abstraction** that assures that the activity terminates in reasonable time and still produces valuable results. Adapt the level of abstraction as is needed for good results.
 - 🌂 Work with **layers:** Some threats can be found and tamed at a high level of abstraction. For some, we can zoom in and dive deeper.
 - 🌂 **Prioritize** and apply project management.
 - 🌂 **Quick wins:** Start with topics from which you can hope for quick success (“low-hanging fruits”).
 - 🌂 **Secure subset:** Determine a subset of the “giant” that has the most valuable features. You may want to threat-model this subset at first, so that you can offer a slightly smaller but still useful and, above all, secure “giant”.
 - 🌂 **Discontinue** parts of the product that are probably insecure and not worth the effort of fixing given their utility.
 - 🌂 Complete the pieces. **Avoid parallel paralysis** (see below).
 - 🌂 **Start today** with techniques like Incremental Threat Modeling or Continuous Threat Modeling (see below, threat model new developments). If relevant work items are already known, mix clarification and implementation efforts. Don’t wait for a big analysis to someday reveal all the priorities.
 - 🌂 Make sure your **threat modeling is effective, efficient and satisfying.** (see various other threat mitigations)
- ▼ 🌂 [1.1.2] **Not catching up**
- The vendor does not spawn threat modeling processes for legacy product parts without a threat model ⇒ This results in ongoing blindness and insecurity (see above).
 - 🌂 Consider catching up. Know the consequences of not catching up (ongoing blindness and insecurity, see above)... Follow a plan (threat



modeling a “giant”, see above), to make sure the activity is a success and the effort is not too high.

Blockers



▼ [1.1.3] Lost momentum

- Threat modeling activities fall asleep. ⇒ The process gets stuck and doesn't produce improvements to the system.
-  Sometimes it is as simple as this: Wake it up again! Invite to the next threat modeling session.
-  Otherwise reveal the true problem and root cause: Often times it has to do with priorities, how busy people are, how much value they see in threat modeling, the contribution of this particular threat modeling activity, management support, or threat modeling that is blocked, inefficient or frustrating. Fix that. (see various suggestions in this document)
-  Or accept that something else is more important right now, if that is the case. Maybe promote later.

▼ [1.1.4] Parallel paralysis

- Threat modelers try to threat model everything all at once. ⇒ This results in a lot of incomplete parallel activities that are stuck somewhere in the middle.
-  Rather complete small and manageable threat model parts with all phases, including implementation, then move on to another part.
-  Choose wisely which teams threat model and how to schedule these teams on threat model parts.

▼ [1.1.5] Overwhelmed teams

- In a comprehensive threat modeling project, the effort is heavily concentrated on certain teams. Meanwhile, other teams can't contribute much.
-  Relieve overwhelmed teams wherever possible.
-  Request that managers boost overwhelmed teams.

- 🌂 Question if threat modeling teams must align with developer teams.
- 🌂 Plan and schedule the activities for a realistic time frame.

▼ 🌂 [1.1.6] **Unclear Priorities**

- Threat modelers start threat modeling activities for parts of the systems that are not that important. Meanwhile, nobody starts activities for important parts of the system.
- 🌂 There should be some sort of coordination which threat modeling activities exist. Process designers should clarify what/who spawns a new threat modeling process and when.

1 🧩 **Threat model new developments [1.2]**

▼ ✨🌂 [1.2.1] **Outdated threat models**

- Developers don't update threat models. ⇒ Threat models get outdated and fail to keep up with new developments. ⇒ New developments may be insecure. We don't know (blindness and insecurity, see above)
- 🌂 Follow the continuous threat modeling mantra: "Threat model every story." For each story, at least consider if it needs new threats and mitigations. If so, update threat models. Integrate that into your process.
- 🌂 Decide if you want to have deferred threat modeling and risk threat model later & (later=never) anti-pattern (see below) or start right away.
- 🌂 Like suggested in continuous threat modeling, have baseline threat models available, where threats and mitigations for new stories can easily be added, at best without having to lazily create whole new threat models.

▼ 🌂 [1.2.2] **Deferred threat modeling / Threat model later & "later=never" anti-pattern**

- When implementing a new feature, developers plan to update the threat model later, then never do it.

- 🌂 Tie threat modeling of changes to the development of enhancements, for instance with a definition of ready / definition of done checkpoint.
- 🌂 The real problem here is not that something is scheduled for later. The problem is that the activity scheduled for later is never done. You may want to fix that instead.

▼ 🌂 [1.2.3] **No threat modeling for new projects**

- Developers create new projects without threat modeling. ⇒ There's blindness and insecurity problems (see above). As the project grows, threat modelers face the problem of threat modeling a "giant" (see above).
- 🌂 When a working process is available and people are educated, use your chance and apply threat modeling early in new projects, so that these projects will be secure by design. ✨ Changes in early designs are cheap and nothing needs repair.
- 🌂 Have standards that require threat modeling for new projects.

▼ ✨🌂 [1.2.4] **Mitigation debt / Security later & "later=never" anti-pattern**

- A new feature is developed, threat model updated, mitigation planned and not yet implemented. The feature is merged. ⇒ The undone mitigation creates technical debt. There is the risk of undone mitigation (see below).
- 🌂 Make sure that you can deliver the mitigation together with the new development. Otherwise, don't merge. Follow the mantra: "It's not done until it's secure".
- 🌂 Or, again, fix your "later=never" problems.

1 🧩 **Scope and represent the system [1.3]**

Threat modelers scope the activity and choose a system representation. Diagrams are often used.

▼ 🌂 [1.3.1] **Too abstract**

- Degree of abstraction is too high. There's too few details. ⇒ Threat discovery doesn't discover meaningful threats.
- ☂ Check system representations if something is too abstract. If so, zoom in.
- ☂ A usual suspect here is having data flows with unknown / implicit power. For each interface, describe in an adequate level of abstraction what it can do.

▼ ✨☁ [1.3.2] **Too detailed**

- Degree of abstraction is too low. There's too much details. ⇒ Threat discovery gets lost and takes forever.
- ☂ Check system representations if something is too detailed. If so, zoom out.
- ☂ Don't try to model "the whole system". "The whole system" is in the source code. We need to zoom out, simplify, condense.
- ☂ Consider splitting into multiple representations or threat modeling activities.

▼ ☁ [1.3.3] **System representation miss**

- Important aspects of the system are forgotten, not modeled and therefore not threat-modeled.
- ☂ Let threat modelers agree in scope beforehand, and only finish the system modeling phase when everyone can look at the diagram/model and agree that yes, that is the system they are modeling, in the right scope

▼ ☁ [1.3.4] **System/Representation mismatch; Garbage in → Garbage out**

- The system representation is too far away from the actual system ⇒ Threat discovery and mitigation planning lack reality respect and produce bad results.
- ☂ No garbage out → No garbage in. Make sure your system representation is adequate and well-informed by people who know the system.

2 "What can go wrong?" Phase

2 🧩 Discover threats [2.1]

📖 Blindness and threat discovery

▼ ✨🌧️ [2.1.1] Blind spot

- Threat modelers miss a crucial threat.
- 🌂 You can never be sure that you see everything. This is because of the asymmetry in security: If being secure is being protected from **all** danger (\forall), then being insecure is not being protected from **some/any** danger (\exists). Perfect security is an ideal goal, but practically unachievable. Don't get obsessed about blind spots. Aim to learn and improve and get better over time. Reveal blind areas. This will significantly improve the process and the security of your system!
- 🌂 Create a positive mindset about threat discovery. If you were not threat modeling, you would find nothing (blindness, see above). So everything you reveal is a win.
- 🌂 Don't let the two things mentioned above deter you from improving. 😊
- 🌂 More than thinking about blind spots (which you can't know), try to reveal blind areas.

▼ ✨🌧️ [2.1.2] Blind area

- Threat modelers miss a whole class of threats.
- 🌂 The Threat Modeling Manifesto names patterns, all of which improve threat modeling practices and reduce blindness: "Systematic approach, informed creativity, varied viewpoints, useful toolkit, theory into practice"
- 🌂 Some usual suspects for blind areas:

- Basic Cyber Security and machines where things are running: Threat modelers sometimes focus on their application only. Can a script kiddie intrude the server with standard tools, just because no one thought about isolation, hardening and making a trust zone from a diagram real?
- Physical security
- Build and deployment
- Insider threats
- Human aspects
- Social engineering
- Mitigation bypasses: Reasons why we introduce Least Privilege / Defense in Depth and think about Game Over Scenarios... What if an attacker overcomes a mitigation? Is that a Game Over Scenario?
- ...
- [I can't tell. I have blind areas about your blind areas. 🙄]
- 🌂 Avoid system representation miss (see above).

▼ ✨☁️ [2.1.3] **Hard threat discovery**





- Threat modelers have a hard time to come up with threats. Especially newbies are overwhelmed.
- 🌂 Let threat modelers threat model. They grow with experience. Establish a culture that supports learning and improvement.
- 🌂 Encourage by conveying the message that everyone can to some extend threat-model intuitively. Convey the image of the "threat modeling muscle" that will be strengthened as we proceed and repeat the exercise.
- 🌂 Know that your first threat models will suck. 🙄 Permission to Suck (kadavy.net). That's okay. We can revisit and improve later. Getting started with room for improvement in quality is still far better than not threat modeling.
- 🌂 Bundle experiences threat modelers with newbies. Let newbies learn from the experienced threat modeler contributions. Let experienced threat

modelers encourage and open the room for newbie contributions.




- 🌂 Offer help and support where appropriate and requested.
- 🌂 Provide simple approachable material for starters. Examples:
 - 4 Question Framework
 - Lightweight methods and prompts, as presented in Shostack + Associates > Shostack + Friends Blog > Fast, Cheap + Good Whitepaper
 - STRIDE
 - Cue cards (Example from ThoughtWorks).
 - Threat Modeling card games (Elevation of Privilege, LINDDUN GO, ...)
 - MITRE CWE Top 25
 - OWASP Top 10
 - Continuous Threat Modeling, especially with it's IFTTT ("if this than that") approach to threat modeling new developments
 - Crypto education from a "Why would I use X" perspective
 - ...
- 🌂 Provide advanced and rich material for people who want to dive deep. Examples:
 - CWE
 - ASVS
 - ATT&CK
 - ...
- 🌂 Have conceptual training.
- 🌂 Have practical training that shows the attacker & defender perspective
- 🌂 Consider using tool- or AI-aided threat discovery
- 🌂 Request help from experienced threat modelers or consultants.

Blockers

▼ [2.1.4] **Stuck in threat discovery / "admiration for the problem"**




- The process gets stuck in threat discovery and does not continue with mitigation planning.
-  Time-box activities.
-  When your threat discovery takes too long, consider zooming out and take a higher level of abstraction. Or cut the activity into multiple parts.
-  Don't identify all threats first, then later think about mitigation, when you observe that your threat discovery takes long. Start mitigating the threats you already discovered. Then move on with more threat discovery.
-  Reflect and be conscious about what you do.

▼ [2.1.5] **Never-ending threat discovery**

- Threat modelers don't know when they are done.
-  Use a structured approach that has a defined end. STRIDE-per-Element is one example
-  When an activity is open-ended, time-box or define enough.
-  Make sure you don't get stuck (stuck, see above)

Relevance and focus

▼ [2.1.6] **Irrelevant threats**

- The threat model gets diluted with irrelevant threats.
-  If you observe that you discover a sequence of irrelevant threats, change your focus and move on to a more valuable topic.
-  Focus on severe threats. You can find some of these by focusing on high impact ("What would be the worst thing that could happen?" / "How does this super-secret thing move through the system?") or high likelihood ("What are the things that everyone can easily do?").
-  Adapt threats to the expected level of the average attacker

- 🌂 Choose wisely whether you note or discard irrelevant threats. Noting them has some value of documenting that you have seen them but considered them irrelevant and helps avoid rewind (rewind, see below)

▼ ✨🌂 [2.1.7] **Loss of big picture**

- The threat model is crowded with lots of threats. Threat modelers lose the big picture, summary or information which threats are important. They still can't tell in a compact manner if the system is secure. What are the key threats and mitigations?
- 🌂 Avoid irrelevance (see above)
- 🌂 Order threats, tag/mark/highlight important threats
- 🌂 Summarize: Cluster threats, end sessions with a summary of lessons learned, ...

▼ 🌂 [2.1.8] **Lack of focus**

- The threat discovery lacks focus and other threats pop up all the time.
- 🌂 Consider reminding threat modelers to focus and follow a structured approach.
- 🌂 If the threat that pops up is important but not related to the current topic, at least take a note and make sure it is not lost. You can discuss and refine the threat later.

▼ 🌂 [2.1.9] **Long excursions**

- Threat modelers get lost in long excursions about "Is this even possible?". They investigate, look at code, try. While this is sometimes fun and creates bitter sweet threat modeling success moments ("This actually works!"), it distracts from threat discovery.
- 🌂 Be conscious about what temptations you follow and when it's better to schedule such an investigation for later.
- 🌂 Sometimes, it's cheaper to just assume that an exploit is possible.
- 🌂 It is a good practice to not discard threats with "But this won't work, because [...]" - rather document the threat and why it won't work.

▼ 🌂 [2.1.10] **Rewind**

- Irrelevant threats pop up again and again in the discussion.
- ☂ Document these with “accepted / no action” and a comment on why. Refer to the note when someone rewinds.

📖 STRIDE (assuming it is used)

▼ ☁ [2.1.11] Mad STRIDE-order

- Threat modelers go nuts trying to apply STRIDE in the STRIDE order.
- ☂ Reorder STRIDE to something that makes more sense, like ITD-SE-R / SE-ITD-R, and iterate.

▼ ☁ [2.1.11b] Categorization quibbles

- Threat modelers spend a lot of time “correctly” classifying threats according to a scheme, like STRIDE.
- ☂ Threat modelers should be aware that the main purpose of such categories is to structure threat discovery and provide a sense of completeness through a structured approach. For a threat at hand, the category is not very important.

📖 Conflicting objectives

▼ ☁ [2.1.12] Hiding the unpleasant

- Threat modelers hide unpleasant threats that come to mind, because they don’t want to deal with the effort of mitigation
- ☂ Relax. Have a positive view of people and assume that everyone wants to do their best work.
- ☂ Design a process and culture that always favors transparent knowledge and honesty over lying, hiding and faking.
- ☂ Make sure your process does not put too much pressure on threat modelers and developers.
- ☂ Provide solutions for unfeasible / high effort mitigations (see below) and undone mitigation threats (see below).

2 Use tool-aided threat discovery [2.2]

▼ [2.2.1] **No-tool miss**

- Manual-only threat discovery misses valuable threats that a tool-aided threat discovery could have found.
- ...?

▼ [2.2.2] **No-AI miss**

- Not using AI misses threats that AI-aided threat discovery could have found.
- ...?

▼ [2.2.2b] **AI hallucinated threats**

- AI aided threat discovery hallucinates threats that are in fact irrelevant.
- ...?

▼ [2.2.3] **Ignored tool**

- A tool is available, but threat modelers / developers don't use it properly or ignore it's outcomes.
- ...?




▼ [2.2.4] **Savior tool misconception**

- Threat modelers have the misconception that a tool-aided threat discovery is complete and has all the answers.
- ...?

▼ [2.2.5] **Drowning in false positives**

- Tool-aided threat discovery creates a lot of false positives. ⇒ False positive fatigue sets in and frustrates threat modelers.
- ...?

▼ [2.2.6] **False false positives**


- Threat modelers reject an issue as a false positive, when it really is a problem. → Bad accept.
- ...?
- ▼  [2.2.7] **Tool leak / AI leak**
 - Use of Tool/AI leaks confidential threat modeling information.
 - ...?
- ▼  [2.2.8] **Tool dictated process**
 - A used tool makes strong specifications for the process and does not really fit the agreed process.
 -  A tool is a helper and shouldn't be telling organizations how they should threat model. The tool needs to adapt to the vendor, not the other way around.
 - ...?

"What are we going to do about it?" Phase

Assess risk / Decide which threats need mitigations [3.1]

Threat modelers judge which threats are worth addressing. This can be done with a likelihood / impact classification scheme or threat ranking. It results in some suggested action, like MUST FIX / SHOULD FIX / COULD FIX / NO NEED TO FIX.

Threats based on risk assessment approach

- ▼  [3.1.1] **No risk assessment**
 - Threat modelers mitigate everything. ⇒ This results in a lot of mitigation effort and all-critical bias (see below).

- 🌂 Focus on severe threats. Either have severity in mind, so that irrelevance (see above) will be avoided. Or apply risk assessment or threat ranking to find out essential threats.

▼ ✨☁️ [3.1.2] Arbitrary risk assessment

- The risk assessment is fuzzy and too much dependent on who assesses the risk and if it is a sunny or cloudy day. ⇒ Decisions can always be questioned, are not reproducible or don't make sense. It is kind of random which threats are addressed.
- 🌂 Introduce a risk assessment methodology with good and consistent judgement that does not suffer from All-acceptable bias or All-critical bias or Insane and untrusted rating scheme (see below).

▼ ☁️ [3.1.3] Reinventing the risk assessment wheel

- Process designers create far too clever rating scheme and don't make use of existing offers ⇒ Design of the risk assessment scheme blocks the threat modeling efforts. The custom scheme may have All-acceptable bias, All-critical bias or suffer from Insane and untrusted rating scheme (see below).
- 🌂 Get inspiration from existing rating schemes. Reuse where possible.

📖 Under-/over- estimating risk

▼ ✨☁️ [3.1.4] Single underestimated risk → Bad accept

- A high risk is judged too low. ⇒ Threat modelers don't mitigate the threat. The system remains vulnerable and threat modelers have a false sense of security. (Mitigation underkill, see below)
- 🌂 Document reasons for accepts. Review accepts. Check if they were justified.
- 🌂 Bad Accept is worse than addressing a risk that doesn't necessarily need to be addressed. In your rating scheme, introduce a slight tendency towards addressing risks.
- 🌂 See mitigation underkill mitigation (see below).

▼ 🌧️ [3.1.5] **All-acceptable bias**

- The rating scheme judges far too many threats acceptable. ⇒ This results in a lot of bad accepts and an insecure system. Threat modelers notice that the scheme rates all-acceptable. They no longer take the results seriously. (Insane and untrusted rating scheme, see below)
- 🌂 Check the rating scheme for sanity. (Insane and untrusted rating scheme, see below)

▼ 🌧️ [3.1.6] **Single overestimated risk**

- A low risk is judged too high. ⇒ This produces mitigations and implementation effort that is not really needed. (Mitigation overkill, see below)
- 🌂 Single overestimated risks are probably not that severe. Vendors may be able to handle some useless effort. Users won't complain about too secure systems, unless there is other problems like usability degradation (see below). Vendors should not allow this to become a structural problem: avoid all-critical bias (see below).
- 🌂 See mitigation overkill mitigation (see below).

▼ ✨🌧️ [3.1.7] **All-critical bias**

- The scheme judges far too many threats critical. ⇒ Mitigation discovery lacks focus and produces all kinds of mitigations, even for irrelevant threats. Developers have huge implementation effort. People question the sanity of the risk assessment scheme, the threat modeling process or even the benefit of threat modeling. Threat modelers notice that the scheme rates all-critical. They no longer take the results seriously. (Insane and untrusted rating scheme, see below)
- 🌂 Check the rating scheme for sanity. (Insane and untrusted rating scheme, see below)

▼ 🌧️ [3.1.8] **Insane and untrusted rating scheme**

- The scheme creates bad risk assessments. ⇒ Threat modelers notice that the scheme rates insane. They no longer take the results seriously. Every

risk assessment is subject to discussion. It is not clear how threats are actually judged and if they require mitigation.

- 🌂 Educate threat modelers to not blindly trust the rating scheme, but also check for themselves if risk assessments are sane.
- 🌂 Allow for feedback. If someone observes that the rating scheme creates insane judgements, document these cases and design something better.
- 🌂 Have an “ejection seat” exception in your process, that conscious human judgement can always overrule an insane rating scheme decision. Document reasons.
- 🌂 Check the rating scheme for sanity. Validate both properties: that severe threats are rated severe (no All-acceptable bias, see above) and unimportant threats are rated no need to fix (no All-critical bias, see above).
- 🌂 In cases where the rating scheme is seriously broken and unusable, stop using it. Fallback to threat modelers’ judgement until a better rating scheme is available.
- 🌂 Embrace the idea of incremental improvement. Apply [Semantic Versioning 2.0.0 | Semantic Versioning \(semver.org\)](#). Have conventions which risk assessments will be repeated when the rating scheme needed an update. Document which version of the rating scheme was used for risk assessments.

📖 Blockers in risk assessment

▼ 🌂 [3.1.9] Too fine-grained risk assessment scheme

- The scheme is too fine-grained. ⇒ Threat modelers have pointless discussions on what exactly to select (“Is this a HIGH or VERY HIGH impact?”). Their choices don’t make a difference in the suggested action.
- 🌂 Use the minimum degree of detail needed for sane judgement.

▼ 🌂 [3.1.10] Never-ending risk assessment

- The risk assessment takes too long. ⇒ Threat modelers get stuck assessing risks for a lot of threats.
- 🌂 Design the risk assessment scheme to be fast, approachable and not too complicated.
- 🌂 Provide threat modelers with “paved roads” so that some discussions on risk assessment have simply already taken place.

3 🧩 Plan mitigations [3.2]

Threat modelers plan countermeasures to address the relevant threats. They aim for a set of mitigations that sufficiently tames each relevant threat and is feasible.

📖 Feasibility

▼ ✨🌂 [3.2.1] Unfeasible / high effort mitigations

- The entirety of mitigations planned creates an implementation effort that is too high and overwhelms the capacities of developers given their priorities. ⇒ Developers get busy. Undone mitigation threats apply (see below). Other value can't be provided.
- 🌂 Reuse mitigations.
- 🌂 Aim for mitigations that can tame a lot of threats.
- 🌂 Consider the implementation effort of mitigations when choosing between alternatives. Favor mitigations that are effective and cheap.
- 🌂 Plan mitigations based on information what is feasible. Involve developers in the process and let them participate in threat modeling.
- 🌂 Threat modeling that is well-informed what is feasible and also understands risk can realistically put all options on the table and consciously decide: Avoid risk, accept risk or choose between different

mitigation alternatives based on effort and protection expected and needed.

- 🌂 Avoid mitigation overkill (see below)

▼ 🌂 [3.2.1b] **Mitigation lost in (risk → priority) translation**

- Threat modelers often judge risk: "critical issue → MUSTFIX". Developers often work guided by priority: "what's next?". They can only handle a certain amount of parallel work in progress, deliver a certain amount of work in a given time and need to integrate all kinds of requests. ⇒ If threat modeling does not translate between risk and priority, threat modelers plan unfeasible mitigations.
- 🌂 Process designers and threat modelers must understand both concerns, integrate and translate between the two ways of working.
- 🌂 Developers should provide threat modelers with a certain capacity to satisfy MUSTFIX for very critical issues.
- 🌂 Threat modelers must be aware that they cannot demand infinite MUSTFIX (unfeasible / high effort mitigations, see above)
- 🌂 Threat modelers should not forget about the incremental nature of system development.
- 🌂 Threat modeling should also be robust to changes in priority and developers not being able to deliver what was planned (undone mitigation, see below).
- 🌂 Development may be able to follow "It's not done until it's secure" for new developments.

📖 **Too weak or too strong mitigations**

▼ ✨🌂 [3.2.2] **Mitigation overkill / "mit Kanonen auf Spatzen schießen"**

- Talk is cheap. Threat modelers can easily add strong and fancy mitigation. ⇒ This results in huge implementation effort.
- 🌂 When suggesting and selecting mitigations, keep in mind that they will have to be implemented for real. 😬

- 🌂 Reach for “good enough” security in both senses of the word: sufficient and not overkill.
- 🌂 Avoid undefined desired level of security (see above)
- 🌂 Avoid All-critical bias in your risk rating (see above), because it will systematically produce mitigation overkill.

▼ ✨🌂 [3.2.3] Mitigation underkill

- A suggested set of mitigations does not really suffice to tame a threat
- 🌂 Over-defend. Apply defense-in-depth. "Failure of single security control is a question of time, failure of security system is a question of design"
- 🌂 Note that this conflicts with the previous goal to not add too many mitigations. Encourage good trade-offs that satisfy both concerns (“good enough” security).
- 🌂 Avoid undefined desired level of security (see above)
- 🌂 Avoid All-acceptable bias in your risk rating (see above), because it will systematically produce mitigation underkill.

📖 Definition of enough

▼ ✨🌂 [3.2.4] Arbitrary definition of enough

- The process lacks a systematic approach how to evaluate if a set of mitigations is good enough. ⇒ The quality is highly dependent on threat modeler decisions and skill. Mitigation overkill and mitigation underkill (see above) occur on a regular basis.
- 🌂 One approach: Apply the risk assessment scheme again, but with the planned mitigations included. Check if the rating left MUST FIX / SHOULD FIX.
- 🌂 Another approach: Estimate protection provided by a mitigation and calculate with risk decreasing factors. Check if the remaining risk is under a certain threshold.
- 🌂 Yet another approach: Let experts decide or review.

▼ 🌧️ [3.2.5] Only satisfying the definition of enough

- If there is a scheme that defines enough mitigation, this scheme may not reward some mitigations even though they are useful. It may reward inferior mitigations. ⇒ Threat modelers choose weaker mitigations only to satisfy the scheme.
- 🌂 Analyze the weaknesses of your definition of enough. Compensate these weaknesses.
- 🌂 Don't forget to let threat modelers judge the remaining risk themselves. Do not solely trust in a definition of enough.

📖 Coming up with mitigations

▼ ✨? 🌧️ [3.2.6] Hard mitigation planning

- Threat modelers have a hard time to come up with mitigations. Especially newbies are overwhelmed.
- 🌂 Help threat modelers understand risk and that it has a likelihood and impact component to it. Consequently, there are likelihood reducer and harm reducer mitigations. These can be explained figuratively as "bicycle locks" and "bicycle helmets".
- 🌂 Teach threat category systems like STRIDE together with a set of standard mitigations that apply for each category. For certain threats, there's obvious default mitigation.
- 🌂 Teach mitigations by example. As part of the training material, process designers can provide threat modelers with a simple threat model of a CRUD example app and present threats with a rich set of mitigation ideas. This can also help demonstrate the ideas of mitigation overkill and mitigation underkill (see above) and convey the idea that depending on the desired level of security, different mitigation approaches can be chosen.
- 🌂 Avoid small toolkit (see below)
- 🌂 The [OWASP Cheat Sheet Series](#) has approachable advice how to do things "the right way".

- 🌂 Comprehensive projects like MITRE CWE and MITRE ATT&CK have mitigation suggestions.
- ...?

▼ 🌂 [3.2.7] **Small toolkit / “If all you have is a hammer, everything looks like a nail”**

- Threat modelers lack knowledge about advanced mitigations. They choose bad or always the same mitigations that are not a good fit for the threat.
- 🌂 Threat modelers benefit from security education, so they come up with good mitigations.
- 🌂 Get familiar with the Explore VS Exploit Concept. If you haven't explored much, explore, don't exploit yet.
- 🌂 Get to know crypto and advanced mitigations, at least from a “what's in it for me?” perspective

▼ 🌂 [3.2.8] **Too much confidence in a particular mitigation**

- Threat modelers overestimate the protection that a particular mitigation provides. In reality, it's easier to overcome the mitigation. ⇒ The actual protection is less than expected. This may result in a threat that is inadequately mitigated. Threat modelers have a false sense of protection.
- 🌂 See mitigation underkill mitigation above. Over-defend. Apply defense-in-depth.

▼ ? 🌂 [3.2.9] **Too little confidence in a particular mitigation**

- Threat modelers underestimate the protection that a particular mitigation provides. ⇒ The mitigation is not chosen in favor of a less adequate fit. (Example: Rejecting encryption because of misconceptions that it is breakable. ⇒ Vulnerable unencrypted system that solely relies on authentication.)
- ...?

▼ 🌂 [3.2.10] **Usability degradation**

- Suggested mitigations don't consider or damage usability.

- 🌂 Let threat modelers consider usability. Aim for solutions that are both usable and secure.
- 🌂 Consider adding usability threat modeling to the mix.

▼ ☁️ [3.2.10b] **Lack of harm reduction**

- Threat modelers only plan mitigations that reduce likelihood, not harm. ⇒ The result is low likelihood disaster scenarios. When these scenarios become real, customers face huge damage and major challenges in incident response.
- 🌂 Assume breach. Plan accordingly.
- 🌂 Threat modelers should also unleash harm reduction. Examples include encryption, anonymization, minimization of data stored, principle of least privilege, detection and response controls, etc.
- 🌂 Especially when a threat is high impact, low likelihood and not sufficiently tamed, harm reduction is probably more promising than reducing likelihood even more.

📖 **Confusion**

▼ ☁️ [3.2.11] **Vague mitigation confusion**

- Threat modelers have an implicit, fuzzy and different understanding of what kind of protection they expect from a mitigation. (Example: Introducing a "Login" without discussing what kind of security level is needed and what would make the login secure.)
- 🌂 Specify mitigations with acceptance criteria. These need not be all the implementation details, but the features they need to provide their security guarantees.

▼ ✨☁️ [3.2.12] **Mixing essentials with ideas**

- Essential mitigations that are required to tame a threat are mixed undistinguishably with nice-to-have ideas that could be implemented some day / maybe. ⇒ Later on, essentials don't get the attention they deserve. Ideas get more attention than they should.

- 🌂 Don't just show ideas how a threat could be tamed. Choose.
- 🌂 Clearly distinguish essential required mitigations and ideas. Specify how you denote both. Keep ideas separate from mitigations.
- 🌂 Mark ideas as "someday / maybe" or "never". Choose wisely, if you want to drop and delete them or document why you decided for something else.
- 🌂 Support the perception that essential mitigation is essential. This will also help avoid undone mitigation (see below).
- 🌂 Have (automatically updated) back-references to threats that help see why a mitigation is needed.

📖 Uncompliance

▼ 🌂 [3.2.13] Uncompliance

- Threat modelers may be satisfied with certain mitigations, but "deviating from security and data management best practices, standards and legislation".
- 🌂 Know what rules apply for your system.
- 🌂 LINDDUN has Uncompliance as it's own threat modeling category.

3 🧩 Implement mitigations [3.3]

▼ ⓘ Praise of implementation / Why is this in-scope?

Threat modelers delegate the implementation to the usual development process. Developers implement mitigations. This step is crucial to actually improving the security of the system.

"Guter Plan. Nix getan." ("Good plan. Nothing done.")

“Es gibt nichts Gutes. Es sei denn, man tut es!” (“There is no good, unless you do it!”)

For some people, threat modeling ends with the plan what to do. They would not consider implementation as part of the threat modeling activity. Also, depending on culture and process, threat modelers may not have the power to influence what work is actually done.




Still, vendors as a whole have to provide implementation of threat modeling mitigations and an effective integration of the processes.

Whatever view you prefer: Make sure, mitigations are implemented!

Watch this funny scene from Izar Tarandach’s talk.

Development process integration

▼ [3.3.1] **Split brain tracking**

- Mitigations are only listed in the threat model, and then forgotten. The threat model is a second source of truth for scheduled work.
-  Create issues for mitigations, like you track any other work that shall be done
-  Integrate processes and tools. Consider which responsibilities the issue tracking system can deliver and what the threat model shall provide.
-  Some people don’t curate up-to-date threat model documents, but operate threat modeling as a process that creates security issues. Consider whether this is sufficient for you.

Undone mitigation

▼ [3.3.2] **Security theater**

- Lack of implementation leaves everyone thinking threat modeling is just some fruitless talk.
- 🌂 Create a strong urge for implementation!
- 🌂 Remember: Implementation is all that connects threat modeling with actual change to the product. Don't detach!
- 🌂 Start to implement! Reduce ambitions, if that helps, and take action. Don't plan comprehensively and forget to act (also known as "death by planning").

▼ ✨🌂 [3.3.3] Undone mitigation

- A planned mitigation is not implemented. It can't be finished on time / for the upcoming release. ⇒ Vendors can't assume that an undone mitigation protects the system from anything. The associated threats are untamed and the system insecure.
- 🌂 Have a trust-worthy development process that gets important issues done in reasonable time.
- 🌂 Plan for success and figure out what is needed to get mitigations done by default.
- 🌂 Avoid unfeasible / high effort mitigations (see above). Let developers have a say in mitigation planning.
- 🌂 Have clarity about which mitigations are essential and which ones are nice-to-have. Decide about rules and follow them, how much known insecurity you accept or if you won't ship insecurity.
- 🌂 When essential mitigations remains undone: Go back the the threat model. Mark unfinished mitigations. Go back to the threats they were trying to tame. Reconsider their mitigation. Maybe find a cheaper mitigation set. Or release the product with some avoidance of that threat (insecure features turned off). Or block the release and finish the mitigation. Or accept the risk for now as a last resort.
- 🌂 Let process designers consider and prepare for these scenarios.

▼ 🌂 [3.3.4] Undone mitigation unnoticed

- System is accidentally released with undone mitigations.
- 🌂 Install activities that assure undone mitigations will be noticed (early), like a mandatory review of the mitigation issues before the release.

▼ 🌂 [3.3.5] **Mitigation rotting**

- Mitigation issue rots underdone somewhere far down in the backlog ⇒ Undone mitigation.
- Tanya Janca has a good video excerpt about this: Forgotten bugs - her #8 DevSecOps Worst Practice - "Don't worry, Tanya, it's in the backlog"
- 🌂 Have a trust-worthy development process that won't forget to finish important issues.
- 🌂 Establish a commitment how security issues are handled. Demand this commitment.
- 🌂 Tag mitigation issues / mark as important / mark high priority / assign target version / assign due date - follow the usual agreed-upon practice of your development process
- 🌂 Let the issues link back to the mitigation in the threat model, so that it can be seen that this an important required mitigation for one or more threats. This discourages accidental degradation.
- 🌂 Educate product owners, deciders and developers about the importance to get them done
- 🌂 Have someone who cares, observes and promotes on a regular basis. Promote issues by reminding, advocating and raising awareness. Use the agreed-upon commitments and markers and issue histories. Understand how prioritization works and can be influenced. Buy cake or flowers if that helps. 😊
- 🌂 Celebrate successes.
- 🌂 Monitor mitigation issues and their rank
- 🌂 Don't mark unimportant stuff important.

▼ 🌂 [3.3.5b] **Meaningless MUSTFIX markers (MMM)**

- The vendor has an agreed-upon scheme how important security work is marked that was supposed to prevent Mitigation rotting. However, everyone has become so used to ignoring these markers, that they don't drive action. ⇒ Mitigation rotting
- ☂ Don't install yet another "really really important" tag.
- ☂ Refresh the commitment. Have someone who promotes.
- ☂ Start small and get something in movement...

▼ ☁ [3.3.6] Mitigation closed

- Mitigation issue is closed without the work being done ⇒ Undone mitigation.
- ☂ Have a trust-worthy development process that won't close issues without considering consequences.
- ☂ Monitor mitigation issues closed
- ☂ When an important issue is closed, come back to the threat model and propose different mitigations. Or reopen the issue.

📖 Utility / Security balance

▼ ☁ [3.3.7] Security eco flame

- There is strong demand for feature implementation and little time/priority for security, like implementing mitigation issues.
- ☂ Work on establishing a mindset that it is not "utility OR security" - we want to build awesome secure stuff, so a certain amount of security is naturally part of the game!
- ☂ If threat modelers avoided mitigation overkill (see above), their mitigations are sane. The links between mitigations and threats help justify why mitigation implementation is important. This helps promote the security work.
- ☂ Avoid undefined desired level of security (see above). When the organization has committed to a certain level of security, threat modelers can demand that commitment.

▼ ☁️ [3.3.8] **Feature eco flame**

- Mitigation implementation effort makes developers busy, so they can't create other value
- ☂️ Reach for "good enough" solutions, avoid high effort mitigations (see above)
- ☂️ Avoid undefined desired level of security (see above). A clear commitment can also help to avoid that security efforts are exaggerated.

📄 **Quality assurance**

▼ ☁️ [3.3.9] **Mitigation implementation error**

- An implementation error causes a mitigation to be ineffective.
- ☂️ Have a trust-worthy development process with quality assurance that knows how to develop in a good quality.
- ☂️ Example: Code Reviews. Tests.
- ☂️ See mitigation underkill mitigation above. Apply defense-in-depth. Don't rely on single security controls.

▼ ☁️ [3.3.10] **Broken mitigation**

- A mitigation implementation breaks with later changes to the product.
- ☂️ Have a trust-worthy development process with quality assurance that assures important things don't break easily.
- ☂️ Example: Have tests assert that your mitigation works. Execute them with each build, before every release, or whatever makes sense.
- ☂️ Conduct penetration tests.

4 "Did we do a good (enough) job?" Phase

4 🧩 **Review threat models and each threat [4.1]**

- 🚧 TODO

4 🧩 Track what is actually tamed [4.2]

- 🚧 TODO

4 🧩 Judge remaining risk [4.3]

- 🚧 TODO

4 🧩 Improve the process with lessons learned [4.4]

▼ ☁️ [4.4.1] **Not reflecting**

- Threat modelers don't reflect ⇒ They don't improve and get rid of their blockers, inefficiencies and frustrations.
- ☂️ Reflect. Do retros.
- ☂️ Capture lessons learned - in terms of content, methodology and team dynamics.

▼ ✨☁️ [4.4.2] **Local learnings**

- Lessons learned are only available for a small team of threat modelers and not shared across teams.
- ☂️ Show successes and fails that help learn. Establish formats where this sharing is possible.
- ☂️ Update the process.


▼ ☁️ [4.4.3] **Detached process design**

- Process designers are detached from threat modelers, so they don't include their learnings in process improvement.

- 🌂 Let process designers watch and talk to threat modelers, so they see where the process needs improvement.
- 🌂 Consider having threat modelers join as process designers.





Overarching aspects

Do collaborative knowledge work in teams with mixed skill level [X.1]

 This section was inspired by [this insightful pre-release conversation about the project at the OWASP #threat-modeling slack channel](#). Special thanks, Matt, Avid and Kim!

Threat Modeling is a "team sport". A lot of threats to the social ecosystem arise from people interacting with problematic traits. This section covers some of these issues that are not special to threat modeling, but may have huge impact on teams and outcomes.

▼ [X.1.1] **Challenging traits in inter knowledge worker communication**



- Knowledge workers collaborating in teams show challenging communication traits that damage the effectiveness, efficiency or satisfaction of single or multiple gatherings.
- **Examples shown as pairs of opposites (X  Y):**
 -  Lonely riding
 -   Wanting to have everyone in the room








- ☁️ Dominating gatherings
vs ☁️ Not contributing
- ☁️ Feeling overly confident in own skill
vs ☁️ Believing you can't do anything
- ☁️ Aiming for perfection
vs ☁️ Acting too sloppy
- ☁️ Rushing
vs ☁️ Doodling
- ☁️ Distracting (acting as the "rabbit hole rabbit")
vs ☁️ Not allowing for insightful excursions
- ☁️ Being too much involved
vs ☁️ Not caring
- ☁️ Ignoring others
vs ☁️ Getting obsessed about what others might think
- ☁️ Fighting / Behaving overly aggressive
vs ☁️ Avoiding conflict / Not standing up for things that are important
- ☁️ Acting SOOO funny
vs ☁️ Never having fun, because the thing you do is SO serious
- ☁️ Suppressing feelings
vs ☁️ Letting emotions rule anything
- ☁️ Only following instructions with total lack of own initiative
vs ☁️ Being out of control
- ☁️ Talking meta all the time, forgetting to get things done
vs ☁️ Not talking meta and making the same mistakes over and over again
- 🌂 Take care of your team(s) and people. Resolve social conflicts.
- 🌂 Meet in the middle. ⚖️ Find the sweet spots.
- 🌂 Test yourself: Which of these traits annoys you the most? You are probably wearing the opposite one.

- 🌂 Be careful to follow the “Stupid is who stupid does” mantra: It’s easier (and less painful) to fix an action you’re doing wrong than fixing who you are.
- 🌂 Bring some tolerance. We’re only humans, after all.


▼ ☁️ [X.1.2] **Challenging traits in learning and interactions with different expertise**


- Collaborating with mixed skill levels, learning and mentor-mentee relationships sometimes show traits that damage the effectiveness, efficiency or satisfaction of learning and performance.
- **Examples shown as pairs of opposites (X vs Y):**
 - ☁️ Not taking into account that people are new and have to learn and grow
vs ☁️ Treating people like newcomers when they have long outgrown it
 - ☁️ Not taking the time to learn (because you are SO busy performing inefficiently)
vs ☁️ Forgetting to perform because you learn all the time and certainly need that one more thing
 - ☁️ Worshiping a guru or mentor and not using your own brain
vs ☁️ Not taking advice and inspiration from more experienced people
 - ☁️ Experienced people ruling the activity, while newbies are residing in spectator mode
vs ☁️ Experienced people denying to contribute, because they want newbies to grow all the time
 - ☁️ Mentors who solve mentees problems when they should better be empowering
vs ☁️ Mentors who don’t help hands-on when mentees are obviously lost and stuck
 - ☁️ Mentees who don’t ask for help
vs ☁️ Mentees who don’t dare to do anything on their own
 - ☁️ Not leaving people alone, always wanting to take part in discussions








  Being absent and unavailable when people really need you

-  Foster a learning and feedback culture.
-  Take into account that people need to learn. Plan how this can be best supported.
-  As a person involved, know your position and your role in the social structure.
-  Resolve social conflicts in mentor-mentee relationships. Reflect about the learning experience.
-  Meet in the middle.  Find the sweet spots.
-  Jeevan Singh -- The Future of Application Security Engineers has good insights how AppSec experts can empower, unleash and then leave alone.

Communicate among threat modelers [X.2]

 *TODO - this section is under construction and only has some ideas.*

 What is really special about inter threat modeler communication and not already covered by the content sections?

-  **"Resistance is futile" misconception - see 0.2.2**
-  **"Won't happen to us" / "We're invulnerable" misconception - see 0.2.3**
- ▼   [X.2.?] **Defending against the devil**   
 - Threat modelers may have naive conceptions about attackers. Different concepts of attackers come with different conceptual baggage. For example, attackers may be framed as evil outside others. In reality, attackers are humans too, they may be insiders, they have their own motivations which are comprehensible from their perspective and may not attack for the sake of defeating humanity.

- ...?
- ▼ ? 🌧️ [X.2.?] **Asking to “Think like an attacker”**
 - Asking to “Think like an attacker” is advice that inexperienced threat modelers may not be able to follow. They have never met attackers. Nor do they know how they attack or what inspires their actions.
 - ...?
- ▼ ? 🌧️ [X.2.?] **Escaping into “One fine day → big rewrite / new product → awesome security” fantasies**
 - Threat modelers or developers sometimes dream of a big rewrite or new product that will someday have awesome security. This deters from small incremental improvements to the current system.
- ▼ ? 🌧️ [X.2.?] **Nothing else matters security**
 - Security people sometimes exaggerate their focus on security and forget that vendors also have to provide utility and that not everybody was hired to secure things.
- ▼ ? 🌧️ [X.2.?] **“No known previous incident = secure” misconception**
 - ...

Call for Feedback

This document embraces the mindset of incremental improvement.

If you have any feedback, please let me know!

What are the Threat Modeling threats you experience as most challenging?

Do you have any better suggestions for mitigations?

Please get in touch:

- <https://hendrik.ewerlin.com/security/>
- E-Mail: hendrik@ewerlin.com
- LinkedIn: [Hendrik Ewerlin | LinkedIn](#)
- OWASP Slack: <https://owasp.slack.com/team/U05EH9V9UG1>
- Threat Modeling Connect Community:
<https://www.threatmodelingconnect.com/members/hewerlin-914>

<https://hendrik.ewerlin.com/security/>