SYNOPSYS®

WHITE PAPER

Threat Modeling, Decoded

Charting the security journey

C. Cummings, S. Figueroa, L. Zhao, and D-M. Gluba

Table of contents

Connected systems are part of the modern world. There is virtually no aspect of life that is not available online, from shopping and booking tickets to dating, banking, and attending medical appointments. And these trends show no sign of stopping. On the contrary, many services are moving to digital-first or digital-only models, and every day, new products—from toasters to autonomous vehicles—are equipped with internet connectivity.

This is also good news for digital criminals. As the number of valuable targets reachable from the internet grows year after year, the digital underworld has professionalized, becoming a fertile ground for new threats and business models that put highly elaborate attacks in the reach of unskilled actors. With state-sponsored threats also on the rise, securing connected systems has become paramount.

To combat digital criminals, we must think like them, swap mindsets to understand what they are after and how they could compromise our systems to achieve it. Threat modeling formalizes this process, giving teams a way to assess the security of their system architecture and, based on the results, decide how to protect what they most value.

In this white paper, we introduce the reader to the world of threat modeling guided by three questions.

- What is the point of threat modeling?
- What does a comprehensive threat model methodology look like?
- In which directions are threat modeling developing?

Why we wrote this document

Threat modeling is a cornerstone of security processes. It provides a systematic way to assess the security of a system at a given point, identifying areas of interest and ways to mitigate risks effectively. However, getting started with threat modeling can be daunting due to the large number of methodologies that exist and the way they all seem to introduce brand-new vocabulary and concepts.

Behind these layers of options hides a pretty straightforward structure, so we wrote this document to shed light on it, introducing the reader to the goals and common elements that constitute a threat model methodology. The document draws examples from the Synopsys methodology, but it also makes it possible to understand what other methodologies try to achieve.

Who needs threat modeling

Every company that deals with information needs security, and threat modeling is an essential component of the security toolkit. Yet different companies may find different approaches that work better for them. Factors such as existing know-how, risk aversion, budget, and external market forces change how companies model threats.

Therefore, it makes sense for companies to draw from different sources to produce an approach that works for them. This paper provides an outline of a threat modeling process that can be filled with different methodologies and approaches, highlighting how the process can benefit from shifting focus as it develops.

Who should read this document

This document is aimed at an audience with a technical mindset, but it does not require deep technical or security-specific knowledge.

That said, we believe this paper could be of interest to you particularly if you

- Are a decision-maker in a software project or in the development of products with a connectivity component (e.g., IoT, embedded, OT)
- Are already familiar with the concept of threat modeling but would like to understand how Synopsys approaches it
- Are intrigued about practical information security

The art of war teaches us to rely not on the likelihood of the enemy's not coming, but on our own readiness to receive him.

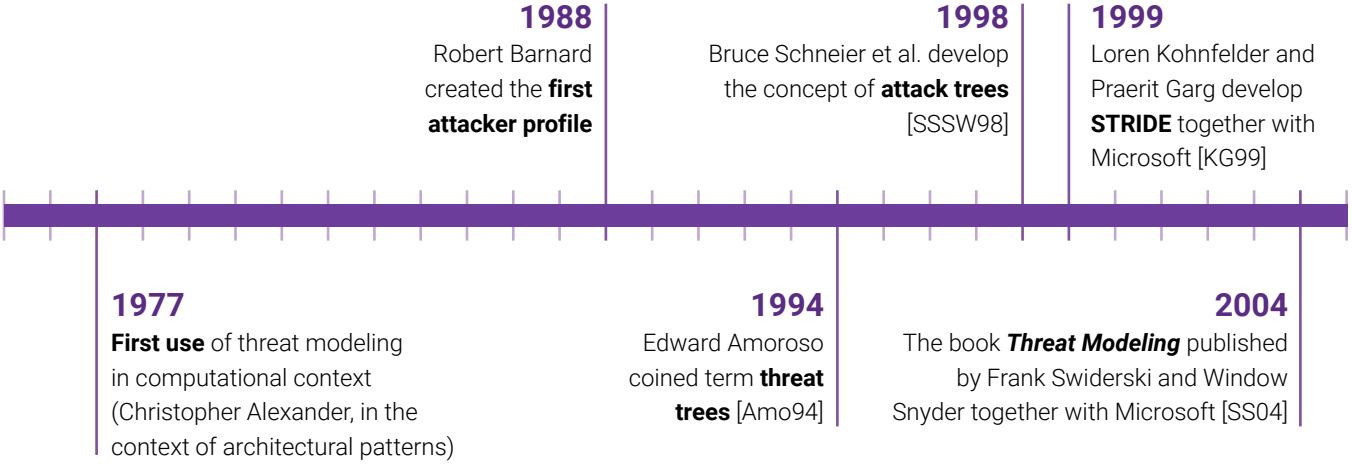
—Sun Tzu

A brief history of threat modeling

Like many other practices and paradigms in information security, threat modeling has military origins [UM15]. Even in Sun Tzu's *The Art of War*, which dates back to the 5th century BCE, we can find vestiges of some of the ideas we will cover in this white paper, such as the "think like an attacker" mindset.

Threat modeling has risen to prominence and continues to evolve because it is the consequence of an arms race started by the advent of connected systems. Humanity keeps using technology to push the boundaries of the possible, reaching into the most sensitive environments and offering insights and control that would baffle Jules Verne or Isaac Asimov, authors famous for their ability to "foresee" the technological future. But sadly, the relentless ingenuity also fuels the world of crime.

The balance between technological development and security is notoriously difficult to strike. A connected or "smart" pacemaker improves a patient's life, for example, but it also puts it at risk in new ways. Threat modeling aims to understand the extent of the risks that a system faces by understanding how it can be attacked. To show how this concept has developed, we hand-picked a few milestones in the history of threat modeling.



Life beyond STRIDE: Four ways to threat model

STRIDE was first introduced in 1999 by Microsoft employees Loren Kohnfelder and Praerit Garg.

Largely due to its simplicity, STRIDE is a widely used way to come up with threats for applications. It looks at the different sorts of mischief an attacker can cause, sorting these actions into neatly defined buckets that form a handily mnemonic acronym.

Spoofing: How can the attacker impersonate our users?

Tampering: How can the attacker modify our data without permission?

Repudiation: How can the attacker hide their tracks?

Information disclosure: How can the attacker leak our secrets?

Denial of service: How can the attacker affect the availability of our system?

Escalation of privilege: How can the attacker gain additional access?

The six categories are understandable yet diverse enough without being overwhelming. The result is a powerful brainstorming tool that is also beginner-friendly. However, STRIDE's role often gets overstated: it's a helpful brain teaser, not a methodology. The scope of a threat modeling methodology is much more comprehensive. It may or may not leverage STRIDE, but that is just one piece of the puzzle.

Not only attacks

It is possible to classify different approaches to threat modeling depending on the aspect of the system they focus on. Each brings benefits and limitations, which is why the best approach is not to worry about which one is best but rather how to combine them into a holistic framework.

Software-centric

Focus on software components, data flows, communication, and trust boundaries.

Asset-centric

Identify valuable elements within the system (assets) and analyze how they are exposed.

Threat-centric

Based on technologies and processes, identify attack vectors and assess their impact.

Attacker-centric

Focus on attackers, from their goals and capabilities to how they interact with the system.

Threat modeling in five steps: The Synopsys approach

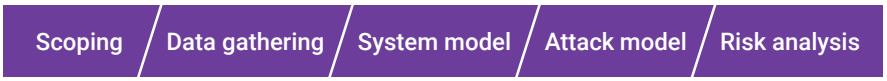
Beware: Simple does not necessarily mean easy. Knowledge, experience, and dedication remain essential.

To most technical people, threat modeling can be daunting. Despite its increasing importance, it's an activity that sounds time-consuming and demanding a lot of domain-specific expertise. An internet search may produce an overwhelming number of methodologies, each with its own concepts (often with overloaded definitions), emphasis, and artifacts.

While the lack of standardization does often get in the way, we argue that most threat models follow a similarly arranged structure that can help eliminate a lot of the noise. With this structure in mind, threat modeling becomes simple.

The basic structure encompasses the five steps.

It is not uncommon that these phases occur in a nonsequential order or are repeated.



A reluctant reader may think, "If it's that simple, then what's so special about this 'Synopsys approach'?" Well, simply put, details matter. In the following sections, we will look at these five steps and the way Synopsys tackles them.

The value of security usually goes unnoticed until it fails, which means that security activities tend to be considered overhead, the stuff that gets in the way of productive work. To counteract this bias, we need an approach that is unlikely to fail but also requires minimal additional effort. This approach must be rigorous and comprehensive but also brisk and easy to understand. Its feedback must be actionable because without a mitigation plan, a vulnerability finding is merely a nitpick.

What is it good for?

Assessing a system's security requires a clear understanding of its purpose and nature: what the system does and what it should not do. Is this a business-critical system or just a minor application? Are we looking at a GUI utility or a sprawling combination of on-premises and cloud systems, mobile applications, embedded devices, and third-party clients? It is important to determine the answers to these questions because one app's security weakness is another app's entire mission.

Similarly relevant for the success of a threat model is the reason to conduct it in the first place. Are there compliance requirements? Is there an internal process to follow? Is the system's security essential to the future of the company? These questions shape the process in subtle ways.

Scoping

The first steps of a threat model are guided by one question: what do we want to protect?

If you have been working on a project for a few months and grown familiar with its architecture, the question *what do we want to protect?* may feel weird: "I want to protect the system against attacks!" sounds like the obvious answer. But there is a lot more nuance to this question than meets the eye.

Scoping is about defining boundaries to guide the subsequent steps.

Computer systems have grown to become complex, interconnected tangles of hardware and software. The combination of underlying infrastructure, off-the-shelf software, and custom and in-house developments makes it hard to determine where one system starts and another ends. Defining the scope is our opening movement to narrow down what we want to protect.

The scope is a conceptual boundary in the system. If a component is inside the scope, we can dissect it, analyze it, and improve it. The scope determines what we consider in a particular analysis. Many methodologies also call this the target of evaluation (TOE). Another aspect are the components that are out of scope but interact with the system. We have much less influence on out-of-scope items and often have to rely on assumptions to fully model their behavior.

Scoping tip 1: Thinking inside the box

It may be hard to draw a solid boundary, especially when the system shares components (e.g., infrastructure, active directory, authentication services) with others, but doing so is essential to attain meaningful outputs.

A good rule of thumb is to look at the general system architecture and draw a single box around the components in scope. No "islands," no "gerrymandering," keep the boundary clean and simple and make sure that the focus of the analysis stays within that box.

Because defining the scope of large or particularly complicated systems is hard, it is often useful to split such a system into complementary threat models. One increasingly popular trend that can help with tackling such a system is that of iterative threat modeling. Instead of going through the five steps once, go over several iterations of the threat model, gradually expanding the scope and refining the threats. Similar to agile development models, this enables faster feedback loops.

Scoping tip 2: Debloating the scope

Another way to define the scope of the system is based on "natural borders" in the system. In other words, you can "de-bloat" your scope by ignoring the items that fall outside of it. These aspects of a system are usually out of scope unless the threat model is specifically focused on them.

- **Infrastructure:** The building blocks the system relies on, but usually cannot influence, like networks, off-the-shelf hardware, operating systems, etc. Note that it is also possible to conduct threat models at infrastructural level (e.g., to assess the security of a given architectural topology or an operating system image).
- **Third-party components:** Elements of the system distributed by third parties like libraries, services, or devices. Note that internal software may count as third party if its definition is not part of the current system (e.g., an SSO service consumed by many applications).
- **Legacy components:** Existing parts of the system that cannot be modified or removed.

This is not to imply that the security of these components does not matter, but usually, when they are scrutinized, they are subject to their own analysis.

Data gathering

Scope definition gives us a vague initial idea of what we want to protect, but the answer is not quite clear yet. If we were cartographers drawing the map of a country, we would similarly start by drawing a vague silhouette. The borders and neighboring countries would be clear, some prominent features would be visible, but most of the territory would still be uncharted.

Hence the next step, data gathering, is one of discovery. We need to dig deep into the scope, then identify the components and their interactions with each other and with their surroundings. Toward the end, we can put our security hat on. What parts of the system are valuable? What undesirable actions can an attacker cause? Are there mechanisms in place to keep this from happening?

Data gathering is about exploration. Threat modelers need to start with a clear route plan but also be able to adjust course (sometimes radically) as they go.

Where to start

No excursion has a straightforward path. Likewise, there are many ways of exploring a system. In a threat model, we focus on the architecture, not the implementation, therefore we would rather get all the information we need from documents such as architecture diagrams, architecture documents, or API interfaces.

Architecture diagrams and documents are great but often unavailable, incomplete, or outdated.

Sadly, real-life documentation often shows a distorted picture of the system—it can, for example, be incomplete or inaccurate, become outdated, deviate from reality, or hide some "temporary" workarounds and exceptions. Just like no cartographer could draw a map only from satellite photos, no matter their quality, we cannot rely only on documentation to analyze a system.

Interviews: Asking around

Interviews are instrumental to fill the gaps left by documentation.

Our solution to the documentation issue is to ask people. At Synopsys, who we ask varies from project to project, but generally speaking, we try to cast as wide a net as possible because systems are usually the product of group efforts, and experience shows us time and again that different group members have different perspectives. By interviewing different roles from technical and business angles, we get an insight into what is common to them and what is important; a closer picture of the actual system. It turns out this approach also has a convenient side effect beyond security: it helps us, and the teams, identify deeply rooted misunderstandings.

The final list of interviewees will depend on variables like the size of the application team, their availability, the timeline of the project, and to a great extent, the emphasis of the analysis (e.g., are hardware experts needed? How much detail about the development process is relevant?). Table 1 shows some of the roles we like to interview and the sorts of questions we like to ask.

ROLE	PURPOSE	EXAMPLE QUESTION(S)
Product owner	Understand business goals, objectives, and key risks	What are the goals of the system?
Software architect	Review system architecture and data flows	What are the main components of the system? How do they communicate with each other?
Developer	Confirm implementation details (of design)	What cryptographic features are used by the application?
Quality assurance	Validate test cases and security testing strategy	What type of testing is being done during development?
Security expert	Understand security controls and threat landscape	What are some known/documented vulnerabilities?
System admin	Review procedures for system management and deployment	How are privileged credentials managed?

Table 1: Sample interview list

Not every project has all these roles (and for some projects, some roles may be missing), but this list hints at the breadth of information that interests us: from business goals and use cases to technologies, constraints, and architectural patterns.

The interview process is a combination of planning and adaptation. Preparation is key: having a rough idea about the target system and bringing along a basic questionnaire to start with is a good way to steer the conversation in the right direction.

But each interview is what it is, a conversation. This is where the informality of the phrase "asking around" hits the nail on its head. More often than not, the interview will go off script as some details become less relevant or new information is revealed. The ability to adjust the plan and come up with meaningful questions on the spot (or to identify the need for additional interviews)—to converse rather than to interrogate—is the key to a successful interview.

Double-checking

It is important to remember that the whole information gathering process is imperfect. All the information from documentation and interviews may be close to the truth because it is provided by knowledgeable stakeholders of the system under review, but it is bound to be plagued by biases, inaccuracies, miscommunication, and many other issues. This is why it is important that we trust but validate.

Validating information independently ensures its plausibility and often helps detect security risks.

A good way to validate the information provided is to contrast the facts against our own research. It is often easy to find best practices, known issues, and security design patterns for common technologies, and by using these sources, we can identify inaccuracies, flaws, and contradictions in the team's answers.

The risk of cryptic claims

One field that repeatedly shows how important it is to scrutinize the team's claims is cryptography.

It is not uncommon for us to hear categorical claims like, "The application provides encryption by default, so we are protected and need not worry about it." These are the claims we need to worry about the most, though, because they hint at a misunderstanding, and cryptography, of all the dimensions of security, is the one where slight misunderstandings have the largest potential to backfire.

Another common example that immediately raises red flags comes when a team says, "We have 64-bit encryption." If this were true, it would suggest the use of old and deprecated cryptography that should not be present in any modern system. But it likely refers to Base64, an encoding scheme that is used in some cryptographic applications but offers no security. To the developers among our readers, this is akin to saying we have ASCII encryption (i.e., it is meaningless). As a common saying among security practitioners goes, "The illusion of security is worse than having no security at all."

Cryptography or not, there is one crucial point: critically questioning the information gathered from interviews and documents is not about challenging the team or trying to catch them in a lie. It is about applying your security know-how and the results of your own research to improve the quality of that information and bringing a new, security-focused perspective to the table.

Documenting assumptions

The value of clearly documenting assumptions is often understated. In a threat model, there are usually factors that are unknown or cannot be influenced under the scope of the analysis. Furthermore, as threat modeling can occur at almost any phase of the software development life cycle (SDLC), it often deals with incomplete or inconsistent designs.

Assumptions allow analysts to acknowledge these sources of uncertainty in a way that is meaningful for the threat model. Typically, they are either pessimistic (e.g., if encryption is optional, it is assumed to be disabled), highly likely (e.g., state-of-the-art technology also relies on the assumption, it is enforced by entities out of the scope of the analysis), or necessary (e.g., servers are trusted).

Showcasing these assumptions allows the people who use the threat model to understand how decisions were made and how risks were analyzed. They also may provide a basis for long-term development (aspects that were covered by an assumption in one iteration could be at the center of the next iteration).

More assumptions

Some examples of assumptions that commonly made in threat models include

- The development environment is trusted.
- The operations/production environment is trusted.
- Communication over HTTP is not encrypted (neither with TLS or otherwise).
- The cloud provider ensures that tenants cannot access each other's managed services.

System model

The **system model** guides the discussion

Once we have formed an impression of the system, what it does, and how it does it, it's time to write it down. Just as in many other aspects of software architecture, having stuff written down (or better, drawn) helps guide the discussion and present results. If a component is missing or a connection is pointing to the wrong target, the system model will help us catch it early.

Our system model gives us diagrams that highlight components, connections, and communication protocols. The information density of diagrams gives us a glimpse at the whole picture and a wealth of detail at the same time.

Software first

Looking at the **software view**, consider some questions. Which components interact with each other? Which technologies and protocols are used? Are there any connections to/from the internet? What trust boundaries exist?

As mentioned previously, at Synopsys, we like to pivot among focus points (attacker, software, assets, etc.) during the analysis. Typically, we start by focusing on software because it often gives us a good view of the goal and the scope of the system, as well as all sorts of security-relevant details like geographic replication, third-party services, roles, and even infrastructure details. Knowing the technologies involved (e.g., cloud, embedded, local application) and the way they are meant to interact allows us to adjust our mindset for later stages.

This level of detail is already enough to make the first opinions about the security of the system. If, say, data is transmitted in plaintext over the internet, we know to be alert, for many other dangerous flaws may be lurking.

Component diagram as base layer

Time to sketch the diagram. There are many sorts of diagrams, and while there is no inherently best diagram notation, it is good to select a notation that is clear (can be understood without a manual of symbols and conventions), well-defined (has an explanation for symbols and conventions), and flexible (can be used to represent monolithic applications, distributed systems, microservices back ends, and everything in between). We need to be able to tell business, technical, and security stories all on the same diagram, clearly and in detail.

One way to build up a component diagram is top-down. We will present this approach in four steps. For better visualization, each step is accompanied by a diagram that refers to a theoretical banking application.

1. We start with the big picture, which should depict the system in scope as a black box, as well as its interactions with the "outside world" (e.g., other systems, users).

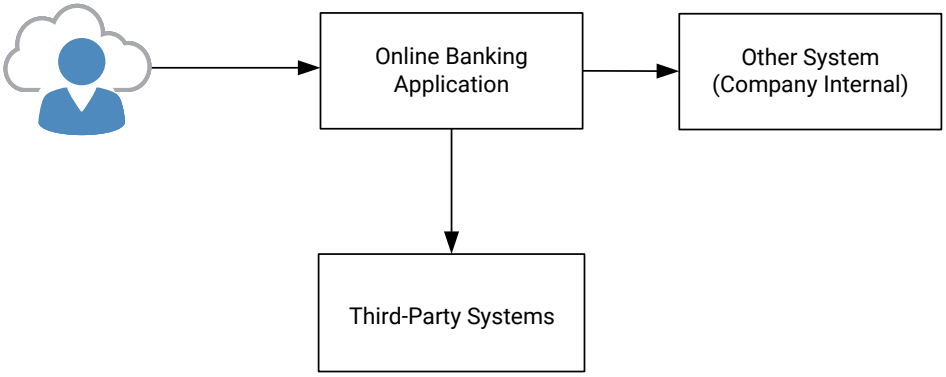


Figure 1: Big picture view

2. Expanding this picture results in a container diagram that reveals clusters of components logically tied together.

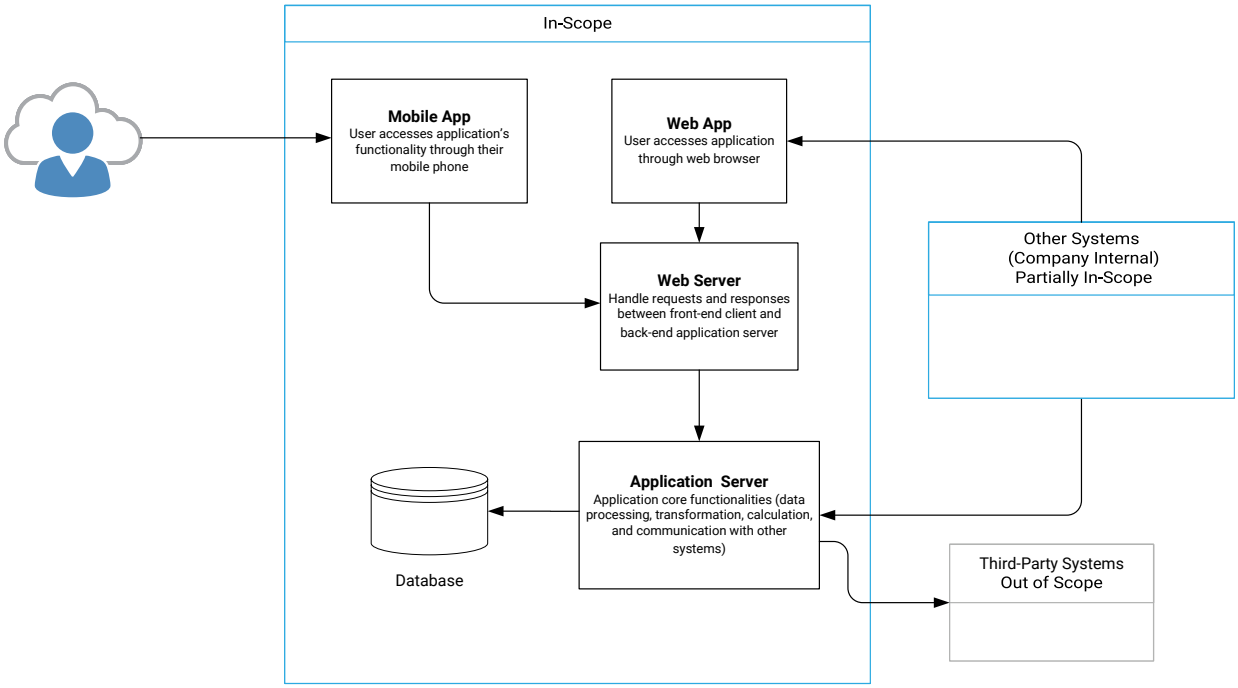


Figure 2: Container view

3. The next level of detail is a component diagram, which explores the insides of each container and the way components communicate with each other. The component diagram should depict connections, protocols, and technologies used in the system.

   The expansion of the application server container in Figure 3 shows its different responsibilities and submodules.

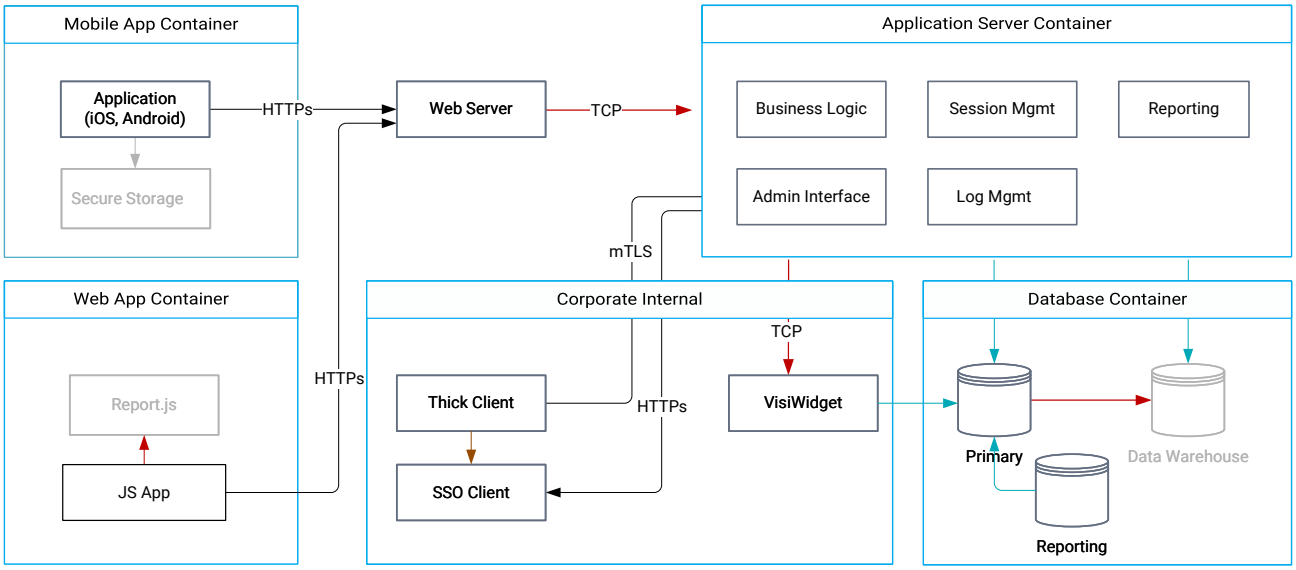


Figure 3: Component view

4. The last step is to add trust boundaries and user context (if missing) to the component diagram. This diagram should show where primary users interact with the system as well as the discovered boundaries (network zones usually make for a good proxy of trust boundaries).

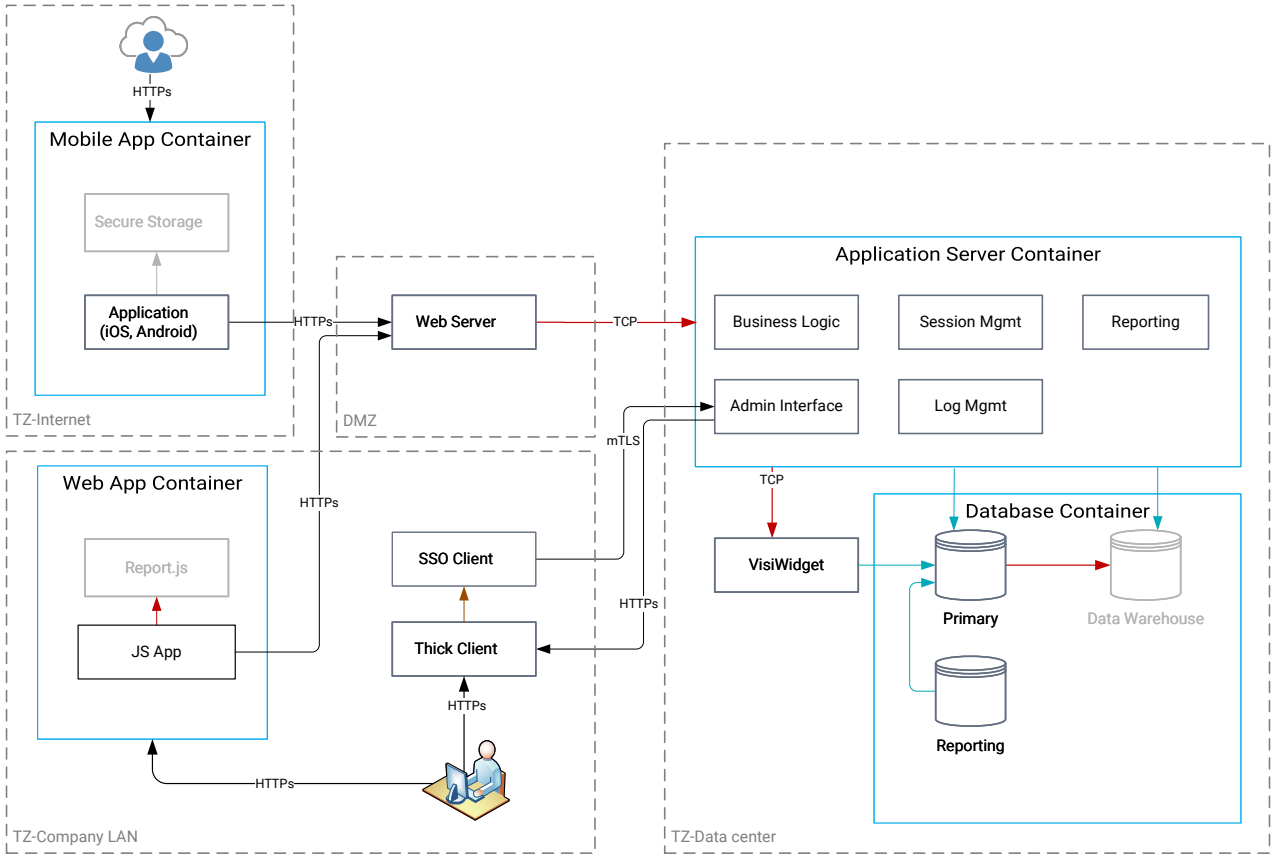


Figure 4: Trust boundaries view

This is an iterative process where we ask ourselves questions about the system and update the diagram accordingly. If we know the answer, we integrate it. If not, we highlight the point as open. We continue asking questions until we feel confident that we know enough about the system.

The stories behind the diagram

The steps may give the impression that building component diagrams is a precise and repeatable approach, but that is not the case. This may not come as a surprise since the source of information for the diagram is the data we gathered, which we know can be incomplete, imprecise, and inaccurate. A lot gets lost in interpretation.

The other aspect of it is the story the diagram tells. Every choice in the diagram, from the content to the style, has an impact on the message it conveys. Perhaps more so than with text, a lot of layers convey a story—the points of arrows, the colors, the layout, everything has a meaning, and it is up to the author to leverage it or to be burdened by it.

The component diagram should tell the following stories:

- What are the components in scope?
- How do the components communicate with each other?
- What are the boundaries between containers? How are they crossed?

The plot thickens when security enters the scene.

Assets

Once the component diagram is sufficiently ripe (it does not need to be finalized just yet), it is time for a perspective shift, a pivot to an asset-centric approach. At this point, our quest to answer the question what do we want to protect? requires us to look at the system through a security-shaded looking glass. In security, we call what we want to protect "assets."

The idea of assets can be found in many industries that deal with value (you may think of money and stocks when you hear it), and it can be as broad as it sounds. Technically, a hardware appliance, an operating system, an off-the-shelf tool, source code, user personal information, and passwords can all be assets if they are of value. However, experience has shown us that, for most software threat models, it is enough to limit the scope to one category: data assets.

Coming up with a list of data assets for a system is arguably the most impactful decision you can make for the whole threat model. If the list is incomplete, you may fail to identify important threats. If the list is too detailed, the threat model may drag on for months without necessarily revealing new information.

A valuable strategy to refine the asset list is to understand why a particular asset is in it. Security properties are a common way to verbalize this reasoning, the most common being the confidentiality, integrity, availability (CIA) triad.

- **Confidentiality** ensures data is only read by intended users.
- **Integrity** ensures data is not forged or tampered with.
- **Availability** ensures data is reachable when needed.

Some assets need the whole triad (e.g., cryptographic material, bank account transactions), but other assets only need one part of it (e.g., open source binaries need to be authentic but are not confidential). And other assets need extended versions of these properties (e.g., logs could be extended but should not be tampered with; firmware images need to be fresh lest an older valid version be deployed to degrade the security of the system).

Security properties help to refine the asset list in a few ways.

- If an asset does not have any security property associated to it, it is not worth protecting (and should be removed from the list).
- If several assets sound similar but have different security properties (e.g., public keys [nonconfidential] and private keys [confidential]), it makes sense to consider them as two assets.
- If some assets have similar security properties and are managed in the same way (e.g., stored in the same location, transmitted over the same sort of network channel), they are good candidates for being merged.

Another criteria to merge or differentiate **assets** is impact, which will be discussed later.

Note that privacy is not usually a matter of security but compliance. The security dimension of privacy is usually covered by confidentiality.

Examples of assets

Coming up with a list of assets may be daunting at first. The following examples of assets and security properties should serve as inspiration, but remember, these buckets and their properties are not set in stone.

- **Personal identifiable information (PII) or other private data:** Confidentiality, integrity, availability
- **Intellectual property (IP):** Confidentiality
- **Configuration:** Integrity, availability
- **Firmware:** Integrity, availability
- **Firmware updates:** Integrity, freshness
- **Logs:** Integrity
- **Cryptographic material:** Confidentiality, integrity, availability

Controls

The system model and the list of assets finally answer the question that has guided us thus far, *what do we want to protect?* Now comes a follow-up question, *how are we protecting it?*

This is where security controls enter the scene. Security controls are the mechanisms the system deploys to protect its assets. Figure 5 shows some ways to classify controls that are useful to understand what a control can be. [Cur20]

Class/effect. How do they mitigate the threat?

- **Preventive:** Eliminate or mitigate the threat directly (e.g., cryptography, secure boot)
- **Detective:** Discover and alert about security incidents (e.g., logging, IDS/IPS)
- **Corrective:** Limit the impact of an attack

Type/nature. Who/what is the target of the control?

- **Technical:** Focus on technology (e.g., cryptography, access control)
- **Administrative:** Focus on people (e.g., background checks, security trainings)
- **Operational:** Focus on policy (e.g., patching, incident response plans)
- **Physical:** Focus on physical access (e.g., key cards, locks, alarms)

Deployment. Where in the system is the control?

- **Network:** For example, firewall, IDS, TLS
- **Server:** For example, full disk encryption
- **Application:** For example, authentication, captcha
- **Database:** For example, database encryption
- **Device:** For example, secure boot
- **Cloud:** For example, access control, log aggregation

Mainly relevant for technical controls

Figure 5: Some ways of classifying controls

In our discussion about assets, we said that threat models benefit from focusing exclusively on data assets. When looking at controls, it is recommended to emphasize technical controls, but it would not be wise to disregard other categories.

Importantly, we should only consider controls that are documented.

Especially in new systems, many expected controls may be missing. This is normal. Resist the impulse to include "obvious" controls that have not actually been built into the system.

And, just as with assets, coming up with a comprehensive list of controls is hard if not done systematically. However, in the case of controls, the answer is easy: go through the list of assets. For each asset, we need to ask how is this asset protected? Many controls protect several assets at once, so as we make it through the list of assets, patterns start to emerge, and we discover fewer and fewer new controls.

Answering the question *how is this asset protected?* does require some creative thinking because it is a many-layered question. Several considerations may help the process of creative discovery.

Hint: The security properties of an asset can give an idea about the sort of controls that can be expected.

- Where is the asset stored? How is the asset protected in storage?
- How is the asset transmitted? How are these communication channels protected?
- Which trust boundaries does the asset cross? How are these boundaries protected?
- Are there copies of the asset leaving the scope? Can these copies affect the information inside the scope?
- Are there security monitoring activities around the system? Are logs informational? Are they monitored? How would the application team react to a vulnerability?
- How is the SDLC process for the target system? Are there activities to review and improve the security of the solution?
- How is cryptographic material handled? Is there any conspicuous risk of eavesdropping or abusing it?

Trust boundaries

Briefly mentioned back in the section, "Component diagram as base layer," trust boundaries depict how trust changes over different regions of the diagram. When a given asset crosses a trust boundary, it faces new threats.

The concept of trust can be subjective and vague. These examples can help by making it more concrete.

- Network segmentation is a natural trust boundary. As data flows from one network segment to the next (e.g., from the internet to a DMZ or from the DMZ to the internal network), it is exposed to different attacks and attackers.
- Many modern computers ship with a device called Trusted Platform Module (TPM), which can store cryptographic material and perform cryptographic operations. The word "trusted" reflects the fact that it defines a series of controls to hinder malicious actors, even those with access to the system, from extracting its valuable secrets. The difference in security controls between the TPM and the rest of the computer can be represented as a trust boundary.

Generally speaking, the more an asset moves across trust boundaries, the more exposed it is to different threats. The system diagram should capture this dynamic.

Security annotations in the diagram

As mentioned at the beginning of the chapter, the system diagram is our main communication tool in the threat model, which means it also needs to convey the security information we have identified. Our notation relies on decorations atop the existing diagram (● assets, ● controls, ● threat agents).

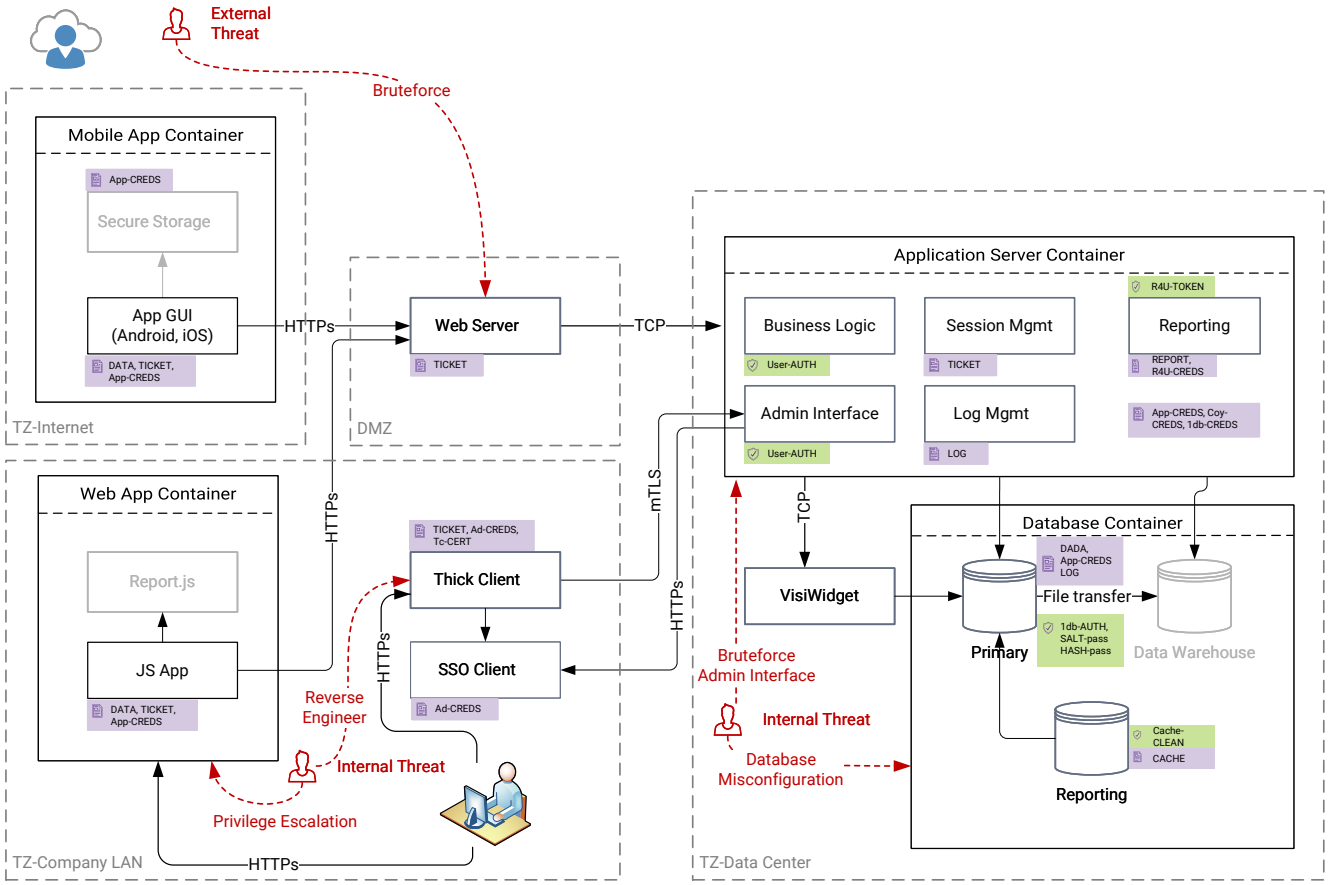


Figure 6: Security threats view

When annotating the diagram, it is important to keep the following in mind:

- We may highlight missing elements—as long as the annotations clearly differentiate from elements present in the system. This should be done sparsely to preserve the communicative power of the diagram.
- Some elements, especially controls, cannot always be mapped to specific points of the diagram (e.g., network segregation, code reviews). In those cases, it is best not to clutter the diagram with them, but they should be documented somewhere (e.g., in a list of all controls).

Wrapping the system model

We have talked about the steps of this process as being sequential, but in practice, going back and forth is not unusual. If we discover a missing component while modeling the system, we can ask the application team further questions or examine additional documentation. During the discovery phase, it is expected that you validate your understanding and uncover new unknowns, so do not be alarmed if you realize later that you have additional questions.

However, before starting with the attack model, the system model should be finalized. A changing system model, which is not uncommon in systems under development, will hurt the usefulness of the threat model, delay the threat model process, or both. Iterative threat modeling is growing in popularity, but even that trend benefits from having a fixed, static scope during each iteration.

Attack model

The sections on data gathering and the system model focused on understanding the system. The application team is the source of expertise and the security team tries to capture that knowledge, and thanks to this interaction, we now know what we want to protect and how it is protected. Now it's time to switch to a different sort of question: is it well protected?

Answering this question requires a shift in the dynamic. As the process moves from describing to assessing, the domain-specific expertise of the threat modeler takes center stage. We need to pivot again, moving now to an attacker- and threat-centric approach.

Is it well protected?

It becomes easier to address the question is it well protected? when we rephrase it using terms we discussed in previous sections. We know the assets have security properties, and we know there are controls that contribute toward these properties. What we're really asking is *are the controls sufficient to protect the security properties of all assets?*

In most cases, this is not a question we can answer with total certainty, but tackling it with the right mindset can give us more helpful answers. This mindset revolves around a popular tenet inside the security community: thinking like an attacker.

When building software, developers face a sort of bias that makes it hard for them to test the system adequately: they expect users to use the system as they would. And most of the time, users do not. Breaking this bias is an extraordinary exercise in empathy and creativity that has grown into several areas of specialization, like user experience (UX) or quality assurance (QA).

Empathy and creativity are also at the core of the thinking like an attacker mindset. The goal is to understand what attackers want to do and the practical (mainly technical) possibilities they can leverage.

Threat actors

Discussions about these topics often use the terms attacker, adversary, threat actor, and threat agent interchangeably.

Threat actors are entities (people or systems) capable of causing a negative impact on an asset. To understand what attackers can do to the system, it often helps to profile them.

We start with four standard attacker profiles based on level of access (internal/external) and authorization level (authorized/unauthorized).

- Internal authorized
- Internal unauthorized
- External authorized
- External unauthorized

The core idea to remember when identifying attacker profiles is that some attacks are only realistic for some attackers. For example

- Remote attackers are dangerous because many of them have varying levels of technical expertise, but they can only reach targets that are connected to the outside world.
- Internal users are in an advantageous position to run an attack because they already have credentials to the system, but they may lack the required know-how or persistence to take an attack to completion.

Attack surfaces

"Attack surface" is a handy phrase for discussing the degree to which a system is exposed. It encompasses all the interfaces that attackers can leverage to achieve their goals in relation to the system. Typically, it refers to the entry point, which means the attacker can poke a hole in the attack surface to get in the system and do a lot of damage once they are inside.

Some examples of different attack surfaces are shown in Figure 7.

Digital, e.g., web server, network channel, file system

Physical, e.g., data center, embedded device components, company premises

Human, e.g., employee with privileges

Figure 7: Classification of attack surface

Threat and attack enumeration

Attacker profiles and attack surfaces are concepts that help guide what is arguably the most time- and effort-consuming task of the threat model: coming up with a list of reasonable and possible threats. There are two main strategies to populate that list.

- Checklist-based approach, where you review a list of possible attacks to determine which ones apply to the system.
- Expert approach, where you brainstorm possible attacks. This approach relies on the experience and creativity of the threat modeler.

The checklist-based approach is limited in nature, as lists may be incomplete and miss contextual hints. An expert approach can be much better, but it relies on the know-how of the reviewers. It is a good idea to combine the two approaches.

Checklist-based approach

The key to taking advantage of a checklist-based approach is easy to express: we need a good checklist. Getting there is an entirely different issue. Creating a good checklist comes with a series of challenges.

- Identifying reputable sources of information
- Consuming and consolidating these sources
- Making the content of the checklist searchable

Identifying reputable sources of information. The information security discipline has become increasingly mature in recent decades, with many organizations becoming reliable producers of security information and making their databases available to the public. Some examples

- MITRE maintains CAPEC and ATT&CK, lists of attack patterns and attack techniques, respectively.
- The Open Web Application Security Project (OWASP) distributes many educational resources, among them the OWASP Top 10 ([OWA21b]), an enormously popular list of the 10 most significant web application security risks. Since the creation of the Top 10 in 2003, OWASP has extended this concept to other fields, such as mobile ([OWA16]), IoT ([OWA18]), and containers (Docker [OWA21a] and Kubernetes [OWA22], both as rolling releases, not standalone documents).
- In addition, OWASP distributes the Application Security Verification Standard (ASVS), a checklist for penetration testing that enumerates attack possibilities for web applications.
- Some industries have also developed industry-specific knowledge bodies, such as AUTO-ISAC in the automotive world.

Consuming and consolidating these sources. The granularity of these sources varies immensely. On one hand, lists like the OWASP Top 10 enumerate very high-level issues to look for, looking at general trends and being updated every few years. Knowing if the system is exposed to one of them requires a deep level of abstraction and understanding of the target system. On the other hand, lists like CVE describe very specific vulnerabilities that may apply to just one version or configuration of a software component. Such a granular source is bound to be incomplete and need continuous maintenance. The more detailed we get about a component, the deeper we understand its security impact, but it is very easy to get overwhelmed by the details.

Consolidating these sources allows a threat modeler to identify potential threats of the given technologies and system components.

Making the content of the checklist searchable. Exploring and indexing these databases is one thing, but being able to do something with them is another. Some of these sources offer ways to query them manually, but there are some limitations to this.

- The filtering capabilities offered tend to be limited. Searching using wildcards, ranges, or attributes like version number or related software libraries is often not possible.
- There is no way to aggregate results from different sources other than to consolidate results manually.
- There is no way to correlate this information with company-internal knowledge.
- The lack of a centralized repository increases the reliance on each threat modeler's process. If a threat modeler does not know of a source, the quality of the threat model could suffer.

Being able to search multiple databases, both to correlate and compare results and to flag outdated or less trustworthy information, provides significant depth and consistency to the threat model. Furthermore, this capability emboldens a trend that has gained traction recently: automating part of the threat model process by integrating it into the SDLC. Access to reliable, structured data may be the defining factor that brings this trend into the mainstream.

Expert approach

The expert approach to detecting weaknesses relies on the experience and creativity of the reviewer to elicit threats that are not covered by a checklist. Many of these threats arise at the seams between components as the way they interact with each other may introduce weaknesses that individual components would not.

Such an experience-based approach is bound to suffer from the reviewer's biases, which cause them to overemphasize some issues and disregard others. To counter the effect of this bias, we recommend following a framework for brainstorming that encourages the reviewer to look at the system from different perspectives.

Figure 8 gleans over the way we use such a framework at Synopsys.

Data. Basic standard controls, like cryptography and access control.

- **Data at rest:** How is data stored (e.g., in databases, local device storage)? How can an attacker read out this information? Would the existing controls thwart these efforts?
- **Cryptography:** Is cryptography used correctly? (Adequate algorithms, parameters, secret management, etc.)
- **Access control:** Are authentication and authorization mechanisms suitable?

Data flow. Trust boundaries, with protection in transit and in different containers.

- **Ingress:** Are incoming messages verified and sanitized?
- **Egress:** Is data only sent to authorized partners?
- **Channel security:** Do the security controls of the network channels protect the communication against attackers?

Platform. Hardware and host systems.

- **Software authenticity:** Can the software running on the system be tampered?
- **Software confidentiality:** Could an attacker learn how the system works (e.g., by getting access to the binaries running on the system)?
- **Hardware security:** How can an attacker gain access to the system using the hardware interfaces (especially highly privileged ones)? Does the solution leverage hardware security features (e.g., TPM)?

Operation. Incident detection and response.

- **Privileged access management:** Does it ensure that proper controls exist to manage access to privileged accounts?
- **Patching:** Does it ensure that processes exist to keep systems up-to-date?

Figure 8: Example of a threat identification framework

Writing findings down

In a way, the next challenge is storytelling. The threat identification process will likely produce an overwhelming number of threats, which is why it is so important to be intentional about how to communicate them. A reader of the threat model report will have questions about the security of the system, and the documentation of the threats found makes or breaks the reader's ability to find answers.

Some generic questions to be expected are

- How many threats are there?
- Are the threats related to each other?
- What threats are the most critical?
- How does one threat affect <<asset>>?
- What is the effect of <<control>>?

Being able to convey which threats matter most and why they do is as important as identifying them in the first place.

Matrices are great for security and compliance audits. They highlight historical threats and applicable controls or lack thereof.

A simple mechanism we use to document threats is the traceability matrix, a table that show information about every threat identified, including mechanisms to associate them with assets, controls, and system components, thus allowing the reader to better grasp (trace back) the context of a threat.

Table 2 shows an example of the fields and layout of a traceability matrix.

THREAT AGENT	ASSET	ATTACK	ATTACK SURFACE	ATTACK GOAL	IMPACT	EXISTING SECURITY CONTROL(S)
Employee	Customer data	Extract information from database	Database	Gain unauthorized access to personal information	High	None
Client	Subscription details	Bypass payment step	Subscription portal	Obtain service without payment	High	Payment token required to finalize subscription

Hint: Full traceability matrices are often better displayed in landscape (wider than taller) format.

Table 2: Example of a traceability matrix

Putting concepts together

A threat describes how an adversary can attack the attack surface of the system to compromise an asset. The adversary has a certain likelihood to succeed, in which case it causes an impact. The combination of these two factors is called risk. The risk of a threat is mitigated by security controls.

Risk analysis

Threat modeling is also known in the industry under other names, such as threat and risk analysis. These names efficiently communicate the central importance of risk as part of the process. It is not enough to understand the threats a system faces, we need a tool to guide our response to these threats—and risk is that tool.

At a high level, the definition of risk in this context is relatively simple: it is a function of two attributes of a threat (attributes that we have mentioned before in this document), likelihood and impact. The details of how likelihood and impact are rated for a particular threat, and how these ratings are aggregated into a risk value, are two of the most significant points of divergence among risk rating methodologies.

Likelihood

Some methodologies prefer to rate attack difficulty instead lest they give the impression that these probabilities are mathematically precise.

A common heuristic is that attackers prefer the path of least resistance: if there are two ways to attack the system, they will prefer the alternative that needs less effort.

The likelihood of a threat is an estimate of how likely the threat is to happen. Many factors can influence threat likelihood.

- **Access**: Who can launch the attack. If a certain attack requires internal and privileged access, then its likelihood is going to be lower. If an attack can be launched by an external user, then the likelihood is going to be higher.
- **Attack sophistication**: Complex attacks are less likely to be executed.
- **Attacker motivation**: Attractive targets will be more likely to be attacked.
- **Controls**: The goal of controls is to make attacks less likely. By their nature, their presence and effectiveness have a direct effect on the likelihood of the attack.
- **Precedent**: Instances of a successful attack may open the door for similar incidents in the future (e.g., the technique becomes reproducible, the attackers keep access into the system).

Rating likelihood

Threat likelihood is not a probability in the mathematical sense: there are too many factors and uncertainties that make it futile to quantify the likelihood of an attack (e.g., "the attacker has a 22% chance of reading information from the hard drive"). Instead, it is common to define categories that make it possible to compare likelihood values without giving the wrong impression of being too precise.

For example, these categories might include

- **Highly likely**: The threat is almost certain to materialize.
- **Likely**: The threat may materialize. Factors such as externally exposed interfaces or missing technical controls make it particularly feasible or attractive.
- **Somewhat likely**: There are some obstacles in the way (e.g., it needs knowledge of a rare technology, reverse engineering, or internal knowledge), but the event can happen.
- **Unlikely**: The threat requires significant malicious effort. It requires, say, valid credentials, significant time, resources, equipment, or a specialized skillset.
- **Highly unlikely**: The threat remains technically possible, but it requires an extremely motivated and skilled attacker (e.g., a well-funded security research team).
- **Impossible**: There is either an accepted assumption or a clearly established reasoning that leads to the conclusion that the attack is not feasible under the current and foreseeable state of technology (e.g., break AES).

Brute-force attack: Repeated attempts to break an authentication mechanism by trying all possible credentials.

To illustrate how this scale can be used, consider the following threat: an online banking application available over the internet requires authentication, and this authentication can be brute-forced. The application is protected against brute-forcing by blocking an IP after three wrong login attempts.

How to rate the likelihood of this attack? Even without knowing many details about the application, we can come to a rough approximation, based on the effect of different factors. For example

▲ Access to the application is through the internet, where it is exposed to all sorts of malicious actors (from users of the application to botnets all over the world looking for targets). The sheer number of adversaries increases the likelihood.

▲ The sophistication level of a brute-force attack is very low. Access to tools and dictionaries of compromised passwords is relatively easy to obtain.

▲ Banking applications present a high-value target and likely increases an attackers motivation, making them a prime target for attack.

▼ Blocking IPs after three login attempts provides a security control that will effectively fend off many automated attacks (reducing likelihood), but such a measure may turn out to be too draconian (locking out legitimate users) and may need to be relaxed in the future.

▲ There is extensive precedent of targeted brute-forcing and dictionary attacks to public-facing applications, which increases the likelihood that it happens again.

It is the task of the reviewer to weigh the impact of these modifiers to determine the likelihood of the threat.

Considering the application is exposed to the internet, requires minimal privileges, easily accessible tools, and resources, and is a high-value target with a known history of successful attacks, it is not unreasonable to consider this threat to be likely or highly likely. However, considering there is a control in place reducing an attacker's ability to automate, the likelihood for a successful attack to occur should be closer to somewhat likely.

Impact

The impact describes the consequences of a threat. This is a way to present the damage of a security breach in terms the decision-makers in an organization can understand and assess. Will the attack cost money? Represent a legal liability? Hurt someone's privacy? Put a life at risk?

Just as with likelihood, the scales for rating impact often vary. Many companies end up developing their own as a way to better capture the factors they value.

These factors can be taken into account to assess the impact of a threat.

- **Extent/number of assets**: How many assets are compromised? Are any of them particularly sensitive (e.g., private information, safety functionality, intellectual property)? Is a recall in order?
- **Extent/sort of compromise**: How are the assets affected? Was secret information leaked? Were important records tampered?
- **Extent/scale**: How much data is affected? Is a single person affected, or was it the entire customer base? Does the attack affect the security of many devices on the field?
- **Availability**: Is the system unavailable to users? For how long? Does it affect a large number of users or different geographic areas?
- **Industry-specific factors**: Depending on the industry, additional factors may need to be considered. In most systems, safety is not a real concern, but it becomes critical in systems with physical impact, such as medical devices or connected cars.

In some scales, the boundaries between categories are clearcut (e.g., if the financial impact is higher than 20% of the company's yearly income). In others, a more flexible qualitative approach is preferred (the team can influence the rating).

Rating impact

As was also the case with likelihood, it is common to cluster the factors that influence the ratings down to a few categories.

- **Critical**: The incident is major and could have severe ramifications, from destabilizing financial impact to the loss of lives.
- **High**: As a result of the incident, the organization may become unable to perform its primary functions for a significant time span or suffer major financial losses.
- **Medium**: The company retains its primary function(s), but some supporting systems may not be operational. The company sustains significant financial damage. The system may cause minor physical damage or injure.
- **Low**: The company is capable of operating almost normally. Low financial damage or operational difficulties may ensue.
- **Minimal**: The effect of the incident is negligible. The company may experience it repeatedly without any meaningful consequence.

Following up on the scenario discussed in the previous section (a banking application that falls victim to a brute-force attack), the impact of this threat can be influenced by these factors.

▼ A successful attack reveals the credentials of a single user. Consequently, the scale of the attack is limited.

▲ The information stolen from the banking application is likely to contain PII and financial information, valuable information even in small volumes.

▲ A successful attack would grant the attacker full read and write access to the compromised account.

▲ As a result of the existing rate limiting control, the account could become temporarily unavailable to the account holder.

It is the task of the reviewer to weigh the impact of these modifiers to determine the real impact of the threat.

Business objectives play a major role in determining the impact of potential security threats. In this case, the scope of the threat is limited, but it impacts sensitive assets in multiple ways, potentially unleashing financial and reputational damage, as well as compliance issues. Due to these aggravating factors, the impact of the threat could be plausibly classified as medium.

Risk severity

The value of a risk, often also called risk severity, is a way to rate and compare risks. Since risk is a function of likelihood and impact, and these two dimensions are often rated in terms of categories, it is also common to define the risk function in terms of categories, a presentation called a risk matrix.

Table 3 shows how a risk matrix may look like based on [NIS12].

LIKELIHOOD	RISK				
Highly likely	Minimal	Low	Medium	High	Critical
Likely	Minimal	Low	Medium	High	Critical
Somewhat likely	Minimal	Low	Medium	Medium	High
Unlikely	Minimal	Low	Low	Low	Medium
Highly unlikely	Minimal	Minimal	Minimal	Low	Low
IMPACT	Minimal	Low	Medium	High	Critical

Table 3: Example of a risk matrix

Risk appetite varies from company to company and industry to industry.

Everyone must independently decide (informed by factors like compliance requirements) the level of risk they are willing to accept.

Severity ratings can then be used to decide how to react to different risks.

Rating risk

- **Critical**: Address immediately or before system go-live.
- **High**: Prioritize mitigation or address before system go-live.
- **Medium**: Document mitigation plan or generate backlog request for systems in development.
- **Low**: Accept or remediate risk. Document mitigation plan.
- **Minimal**: Accept risk. Document mitigation plan.

Where do we go from here?

Many computer games use the concept of "fog of war." At the start of the game, the map only shows the player's immediate surroundings and a few items of interest. The undiscovered vastness around is, as it were, remains foggy, but as the game advances, the player discovers new areas and new resources. The game is not beaten, but the player becomes able to do much more.

We hope this paper helped lift the fog around threat modeling. Threat modeling is not a trivial activity, and it does require time, expertise, collaboration, and a well-defined process to yield valuable resources. But knowing the five steps will help you understand the mindset, intentions, and different approaches behind different threat modeling approaches.

We started by drawing a fence around the system to determine what we want to discuss and what we do not (scoping). Then we looked at how to discover information about the system (data gathering) and to organize this information into a description (system model), which, summarized in the form of diagrams, allows everyone involved to discuss the system on the same terms. Finally, we explored how to identify the threats looming over the system (attack model) and how to evaluate the seriousness of each threat (risk analysis).

This exploration of the five steps is a tool for you. As we mentioned earlier, threat modeling looks somewhat differently from company to company, so like the player in the game, it is up to you to customize your approach in a way that brings you comfort and results.

Some questions you can consider

- How would you incorporate the five steps in your existing development processes?
- Should you do a threat model once, or is there value in an iterative and incremental approach?
- How can the mindset described in this document enhance your approach to security? Some ideas
  - Approach the system with different focus at different times.
  - Tweak the breadth and width of your analysis with careful scoping and definition of assumptions.
  - Think like an attacker to identify threats and weaknesses.
  - Focus on the parts of the system you value most.
  - Rely on expertise (internal and external), especially knowledge bases.
  - Risk and its components (likelihood and impact) are not necessarily a mathematical function. Being able to compare results roughly is often enough, and these scales should reflect the company's risk appetite.

This paper opened the doors of threat modeling, the entry point to a long discussion. There are many promising trends developing at the moment, and we look forward to sharing with you our views on the future and to hearing your perspective about it.

Glossary

adversary (also attacker or threat agent) Actor (human or system) that conducts an attack.

approach A way to apply a framework.

asset Any resource worth protecting (e.g., data, functionality, services, people, physical resources). IT security analyses typically focus on data assets.

attack An undesired event (often intentional) that affects the security of one or more assets.

attack surface The collection of components and interfaces that an adversary could use to conduct an attack.

attack tree An acyclic graph that describes different ways to perform an attack. While the original concept of attack trees (as the name suggests) proposed a tree-like structure, some implementations rely on cross-references to capture technical steps that are part of more than one attack. (Note: the concept of attack trees was adapted from a safety-related technique called fault analysis trees.)

attack vector A sequence of steps to conduct an attack.

component Any element of a system (e.g., technology, processes, users, administrators) that stores, processes, or relays data.

exploit Action that leverages a vulnerability in the system to advance an attack.

framework High-level structure that defines what a process should achieve, often in terms of milestones and deliverables. As a framework is often loose in the details, it may result in different ways of implementing it. We call these ways "approaches."

freshness Quality of data that describes how current it is. Authentic data is fresh when it is created, but it may become stale (non-fresh) over time (e.g., when new data is created or after a time window has passed). See also: replay attack.

impact The (negative) consequences of an attack.

likelihood The approximate probability that an attack will occur.

replay attack An attack whereby an attacker captures a valid message or piece of data, then reuses it at a later point to repeat its effect. In embedded devices, downgrade attacks follow a similar strategy to force the device to boot older (potentially less secure) versions of the firmware. Ensuring the freshness of messages and data is a common mitigation strategy against these attacks.

risk A function of the likelihood and impact of an attack that enables the comparison of different attacks.

scope The components and aspects of the system that will be analyzed.

security control A technology, process, or policy that removes, counters, or mitigates a risk.

threat A potential attack against the system.

threat agent See *adversary*.

traceability matrix A tabular artifact designed to document threats and their relation with the system.

trust boundaries Conceptual "borders" that separates zones of the system with different trust levels (e.g., where data leaves the company network and reaches the internet, or where some information leaves a sandboxed environment).

vulnerability Weakness in a system that can be abused to advance an attack. This is comparable to the concept of a bug in software development, except that it requires malicious intent to be triggered.

Acronyms

ASVS Application Security Verification Standard

C4 context, containers, components, and code

CIA confidentiality, integrity, availability

CC common criteria

CVE Common Vulnerabilities and Exposures

EDA electronic design automation

IP intellectual property

OWASP Open Web Application Security Project

PII personal identifiable information

PM project manager

PO product owner

SDLC software development life cycle

TOE target of evaluation

TPM Trusted Platform Module

References

[Amo94] Edward G Amoroso, "Fundamentals of Computer Security Technology," Prentice-Hall, Inc., 1994.

[Cur20] Blake Curtis, "What is the difference between requirements and controls?" ISACA Now Blog, 2020; https://www.isaca.org/re-sources/news-and-trends/isaca-now-blog/2020/what-is-the-difference-between-requirements-and-controls.

[KG99] Loren Kohnfelder and Praerit Garg, "The Threats to Our Products," Microsoft Corporation, 1999.

[NIS12] US NIST, "NIST Special Publication 800-30, Guide for Conducting Risk Assessments," 2012.

[OWA16] OWASP, "OWASP Mobile Top 10," 2016; https://owasp.org/ www-project-mobile-top-10/. [Accessed 13-01-2023].

[OWA18] OWASP, "OWASP Internet of Things Top 10," 2018; https://owasp.org/ www-pdf-archive/OWASP-IoT-Top-10-2018-final.pdf. [Accessed 13-01-2023].

[OWA21a] OWASP, "OWASP Docker Top 10," 2021; https://github.com/OWASP/Docker-Security. [Accessed 13-01-2023].

[OWA21b] OWASP, "OWASP Top 10:2021," 2021; https://owasp.org/Top10/. [Accessed 13-01-2023].

[OWA22] OWASP, "OWASP Kubernetes Top 10," 2022; https://owasp.org/ www-project-kubernetes-top-ten/. [Accessed 13-01-2023].

[SS04] Frank Swiderski and Window Snyder, "Threat Modeling," Microsoft Press, 2004.

[SSSW98] Chris Salter, et al., "Toward a secure system engineering methodology," in Proceedings of the 1998 Workshop on New Security Paradigms, 1998, pp. 2–10.

[UM15] Tony Uceda-Velez and Marco M. Morana, "Risk Centric Threat Modeling: Process for Attack Simulation and Threat Analysis," John Wiley & Sons, 2015.

SYNOPSYS®

The Synopsys difference

Synopsys provides integrated solutions that transform the way you build and deliver software, accelerating innovation while addressing business risk. With Synopsys, your developers can secure code as fast as they write it. Your development and DevSecOps teams can automate testing within development pipelines without compromising velocity. And your security teams can proactively manage risk and focus remediation efforts on what matters most to your organization. Our unmatched expertise helps you plan and execute any security initiative. Only Synopsys offers everything you need to build trust in your software.

For more information, go to www.synopsys.com/software.

Synopsys, Inc.
690 E Middlefield Road
Mountain View, CA 94043 USA

Contact us:
U.S. Sales: 800.873.8193
International Sales: +1 415.321.5237
Email: sig-info@synopsys.com