[AWS Security Blog](https://aws.amazon.com/blogs/security/)

How to approach threat modeling

by Darran Boyd | on 11 JAN 2021 | in [AWS Well-Architected](https://aws.amazon.com/blogs/security/category/aws-well-architected/), [Intermediate (200)](https://aws.amazon.com/blogs/security/category/learning-levels/intermediate-200/), [Security, Identity, & Compliance](https://aws.amazon.com/blogs/security/category/security-identity-compliance/) | [Permalink](https://aws.amazon.com/blogs/security/how-to-approach-threat-modeling/) | [Comments](https://commenting.awsblogs.com/embed.html?disqus_shortname=aws-security-blog&disqus_identifier=18119&disqus_title=How+to+approach+threat+modeling&disqus_url=https://aws.amazon.com/blogs/security/how-to-approach-threat-modeling/) | [Share](https://aws.amazon.com/blogs/security/how-to-approach-threat-modeling/)

In this post, I’ll provide my tips on how to integrate threat modeling into your organization’s application development lifecycle. There are many great guides on how to perform the procedural parts of threat modeling, and I’ll briefly touch on these and their methodologies. However, the main aim of this post is to augment the existing guidance with some additional tips on how to handle the people and process components of your threat modeling approach, which in my experience goes a long way to improving the security outcomes, security ownership, speed to market, and general happiness of all involved. Furthermore, I’ll also provide some guidance specific to when you’re using [Amazon Web Services (AWS)](https://aws.amazon.com/products/).

Let’s start with a primer on threat modeling.

**Why use threat modeling**

IT systems are complex, and are becoming increasingly more complex and capable over time, delivering more business value and increased customer satisfaction and engagement. This means that IT design decisions need to account for an ever-increasing number of use cases, and be made in a way that mitigates potential security threats that may lead to business-impacting outcomes, including unauthorized access to data, denial of service, and resource misuse.

This complexity and number of use-case permutations typically makes it ineffective to use unstructured approaches to find and mitigate threats. Instead, you need a systematic approach to enumerate the potential threats to the workload, and to devise mitigations and prioritize these mitigations to make sure that the limited resources of your organization have the maximum impact in improving the overall security posture of the workload. Threat modeling is designed to provide this systematic approach, with the aim of finding and addressing issues early in the design process, when the mitigations have a low relative cost compared to later in the lifecycle.

The [AWS Well-Architected Framework](https://aws.amazon.com/architecture/well-architected/) calls out threat modeling as a specific best practice within the [Security Pillar](https://wa.aws.amazon.com/wat.pillar.security.en.html), under the area of [foundational security](https://wa.aws.amazon.com/wat.pillar.security.en.html#sec.security), under the question [SEC 1: How do you securely operate your workload?](https://wa.aws.amazon.com/wat.question.SEC_1.en.html):

“Identify and prioritize risks using a threat model: Use a threat model to identify and maintain an up-to-date register of potential threats. Prioritize your threats and adapt your security controls to prevent, detect, and respond. Revisit and maintain this in the context of the evolving security landscape.”

Threat modeling is most effective when done at the workload (or workload feature) level, in order to ensure that all context is available for assessment. AWS Well-Architected defines a [workload](https://wa.aws.amazon.com/wat.concept.workload.en.html) as:

“A set of components that together deliver business value. The workload is usually the level of detail that business and technology leaders communicate about. Examples of workloads are marketing websites, e-commerce websites, the back-ends for a mobile app, analytic platforms, etc. Workloads vary in levels of architectural complexity, from static websites to architectures with multiple data stores and many components.”

**The core steps of threat modeling**

In my experience, all threat modeling approaches are similar; at a high level, they follow these broad steps:

1. Identify assets, actors, entry points, components, use cases, and trust levels, and include these in a design diagram.
2. Identify a list of threats.
3. Per threat, identify mitigations, which may include security control implementations.
4. Create and review a risk matrix to determine if the threat is adequately mitigated.

To go deeper into the general practices associated with these steps, I would suggest that you read the [SAFECode Tactical Threat Modeling](https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf) whitepaper and the [Open Web Application Security Project (OWASP) Threat Modeling Cheat Sheet](https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html). These guides are great resources for you to consider when adopting a particular approach. They also reference a number of tools and methodologies that are helpful to accelerate the threat modeling process, including creating threat model diagrams with the OWASP [Threat Dragon project](https://owasp.org/www-project-threat-dragon/) and determining possible threats with the [OWASP Top 10](https://owasp.org/www-project-top-ten/), [OWASP Application Security Verification Standard (ASVS)](https://owasp.org/www-project-application-security-verification-standard/) and [STRIDE](https://en.wikipedia.org/wiki/STRIDE_%28security%29). You may choose to adopt some combination of these, or create your own.

**When to do threat modeling**

Threat modeling is a design-time activity. It’s typical that during the design phase you would go beyond creating a diagram of your architecture, and that you may also be building in a non-production environment—and these activities are performed to inform and develop your production design. Because threat modeling is a design-time activity, it occurs *before* code review, code analysis (static or dynamic), and penetration testing; these all come later in the security lifecycle.

Always consider potential threats when designing your workload from the earliest phases—typically when people are still on the whiteboard (whether physical or virtual). Threat modeling should be performed during the design phase of a given workload feature or feature change, as these changes may introduce new threats that require you to update your threat model.

**Threat modeling tips**

Ultimately, threat modeling requires thought, brainstorming, collaboration, and communication. The aim is to bridge the gap between application development, operations, business, and security. There is no shortcut to success. However, there are things I’ve observed that have meaningful impacts on the adoption and success of threat modeling—I’ll be covering these areas in the following sections.

**1. Assemble the right team**

Threat modeling is a “team sport,” because it requires the knowledge and skill set of a diverse team where all inputs can be viewed as equal in value. For all listed personas in this section, the suggested mindset is to start from your end-customers’ expectations, and work backwards. Think about what your customers expect from this workload or workload feature, both in terms of its security properties and maintaining a balance of functionality and usability.

I recommend that the following perspectives be covered by the team, noting that a single individual can bring more than one of these perspectives to the table:

The Business persona – First, to keep things grounded, you’ll want someone who represents the business outcomes of the workload or feature that is part of the threat modeling process. This person should have an intimate understanding of the functional and non-functional requirements of the workload—and their job is to make sure that these requirements aren’t unduly impacted by any proposed mitigations to address threats. Meaning that if a proposed security control (that is, mitigation) renders an application requirement unusable or overly degraded, then further work is required to come to the right balance of security and functionality.

The Developer persona – This is someone who understands the current proposed design for the workload feature, and has had the most depth of involvement in the design decisions made to date. They were involved in design brainstorming or whiteboarding sessions leading up to this point, when they would typically have been thinking about threats to the design and possible mitigations to include. In cases where you are not developing your own in-house application (e.g. [COTS](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) applications) you would bring in the internal application owner.

The Adversary persona – Next, you need someone to play the role of the adversary. The aim of this persona is to put themselves in the shoes of an attacker, and to critically review the workload design and look for ways to take advantage of a design flaw in the workload to achieve a particular objective (for example, unauthorized sharing of data). The “attacks” they perform are a mental exercise, not actual hands-on-keyboard exploitation. If your organization has a so-called [Red Team](https://en.wikipedia.org/wiki/Red_team), then they could be a great fit for this role; if not, you may want to have one or more members of your security operations or engineering team play this role. Or alternately, bring in a third party who is specialized in this area.

The Defender persona – Then, you need someone to play the role of the defender. The aim of this persona is to see the possible “attacks” designed by the adversary persona as potential threats, and to devise security controls that mitigate the threats. This persona also evaluates whether the possible mitigations are reasonably manageable in terms of on-going operational support, monitoring, and incident response.

The AppSec SME persona – The Application Security (AppSec) subject matter expert (SME) persona should be the most familiar with the threat modeling process and discussion moderation methods, and should have a depth of IT security knowledge and experience. Discussion moderation is crucial for the overall exercise process to make sure that the overall objectives of the process are kept on-track, and that the appropriate balance between security and delivery of the customer outcome is maintained. Ultimately, it’s this persona who endorses the threat model and advises the scope of the actions beyond the threat modeling exercise—for example, penetration testing scope.

**2. Have a consistent approach**

In the earlier section, I listed some of the popular threat modeling approaches, and which one you select is not as important as using it consistently both within and across your teams.

By using a consistent approach and format, teams can move faster and estimate effort more accurately. Individuals can learn from examples, by looking at threat models developed by other team members or other teams—saving them from having to start from scratch.

When your team estimates the effort and time required to create a threat model, the experience and time taken from previous threat models can be used to provide more accurate estimations of delivery timelines.

Beyond using a consistent approach and format, consistency in the granularity and relevance of the threats being modeled is key. Later in this post I describe a recommendation for creating a catalog of threats for reuse across your organization.

Finally, and importantly, this approach allows for scalability: if a given workload feature that’s undergoing a threat modeling exercise is using components that have an existing threat model, then the threat model (or individual security controls) of those components can be reused. With this approach, you can effectively take a dependency on a component’s existing threat model, and build on that model, eliminating re-work.

**3. Align to the software delivery methodology**

Your application development teams already have a particular workflow and delivery style. These days, Agile-style delivery is most popular. Ensure that the approach you take for threat modeling integrates well with both your delivery methodology and your tools.

Just like for any other deliverable, capture the user stories related to threat modeling as part of the workload feature’s sprint, epic, or backlog.

**4. Use existing workflow tooling**

Your application development teams are already using a suite of tools to support their delivery methodology. This would typically include collaboration tools for documentation (for example, a team wiki), and an issue-tracking tool to track work products through the software development lifecycle. Aim to use these same tools as part of your security review and threat modeling workflow.

Existing workflow tools can provide a single place to provide and view feedback, assign actions, and view the overall status of the threat modeling deliverables of the workload feature. Being part of the workflow reduces the friction of getting the project done and allows threat modeling to become as commonplace as unit testing, QA testing, or other typical steps of the workflow.

By using typical workflow tools, team members working on creating and reviewing the threat model can work asynchronously. For example, when the threat model reviewer adds feedback, the author is notified, and then the author can address the feedback when they have time, without having to set aside dedicated time for a meeting. Also, this allows the AppSec SME to more effectively work across multiple threat model reviews that they may be engaged in.

Having a consistent approach and language as described earlier is an important prerequisite to make this asynchronous process feasible, so that each participant can read and understand the threat model without having to re-learn the correct interpretation each time.

**5. Break the workload down into smaller parts**

It’s advisable to decompose (break down) the workload into features and perform the threat modeling exercise at the feature level, rather than create a single threat model for an entire workload. This approach has a number of key benefits:

1. Having smaller chunks of work allows more granular tracking of progress, which aligns well with development teams that are following Agile-style delivery, and gives leadership a constant view of progress.
2. This approach tends to create threat models that are more detailed, which results in more findings being identified.
3. Decomposing also opens up the opportunity for the threat model to be reused as a dependency for other workload features that use the same components.
4. By considering threat mitigations for each component, at the overall workload level this means that a single threat may have multiple mitigations, resulting in an improved resilience against those threats.
5. Issues with a single threat model, for example a critical threat which is not yet mitigated, does not become launch blocking for the entire workload, but rather just for the individual feature.

The question then becomes, how far should you decompose the workload?

As a general rule, in order to create a threat model, the following context is required, at a minimum:

* One asset. For example, credentials, customer records, and so on.
* One entry point. For example, [Amazon API Gateway](https://aws.amazon.com/api-gateway/) REST API deployment.
* Two components. For example, a web browser and an API Gateway REST API; or API Gateway and an [AWS Lambda](https://aws.amazon.com/lambda/) function.

Creating a threat model for a given AWS service (for example, API Gateway) in isolation wouldn’t fully meet this criteria—given that the service is a single component, there is no movement of the data from one component to another. Furthermore, the context of all the possible use cases of the service within a workload isn’t known, so you can’t comprehensively derive the threats and mitigations. AWS performs threat modeling of the multiple features that make up a given AWS service. Therefore, for your workload feature that leverages a given AWS service, you wouldn’t need to threat model the AWS service, but instead consider the various AWS service configuration options and your own workload-specific mitigations when you look to mitigate the threats you’ve identified. I go into more depth on this in the “Identify and evaluate mitigations” section, where I go into the concept of baseline security controls.

**6. Distribute ownership**

Having a central person or department responsible for creation of threat models doesn’t work in the long run. These central entities become bottlenecks and can only scale up with additional head count. Furthermore, centralized ownership doesn’t empower those who are actually designing and implementing your workload features.

Instead, what scales well is distributed ownership of threat model creation by the team that is responsible for designing and implementing each workload feature. Distributed ownership scales and drives behavior change, because now the application teams are in control, and importantly they’re taking security learnings from the threat modeling process and putting those learnings into their next feature release, and therefore constantly improving the security of their workload and features.

This creates the opportunity for the AppSec SME (or team) to effectively play the moderator and security advisor role to all the various application teams in your organization. The AppSec SME will be in a position to drive consistency, adoption, and communication, and to set and raise the security bar among teams.

**7. Identify entry points**

When you look to identify entry points for AWS services that are components within your overall threat model, it’s important to understand that, depending on the type of AWS service, the entry points may vary based on the architecture of the workload feature included in the scope of the threat model.

For example, with [Amazon Simple Storage Service (Amazon S3)](https://aws.amazon.com/s3/), the possible types of entry-points to an S3 [bucket](https://docs.aws.amazon.com/AmazonS3/latest/dev/UsingBucket.html) are limited to what is exposed through the [Amazon S3 API](https://docs.aws.amazon.com/AmazonS3/latest/API/API_Operations.html), and the service doesn’t offer the capability for you, as a customer, to create additional types of entry points. In this Amazon S3 example, as a customer you make choices about how these existing types of endpoints are exposed—including whether the bucket is private or publicly accessible.

On the other end of the spectrum, [Amazon Elastic Compute Cloud (Amazon EC2)](https://aws.amazon.com/ec2/) allows customers to create additional types of entry-points to EC2 instances (for example, your application API), besides the entry-point types that are provided by the [Amazon EC2 API](https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_Operations.html) and those native to the operating system running on the EC2 instance (for example, SSH or RDP).

Therefore, make sure that you’re applying the entry points that are specific to the workload feature, in additional to the native endpoints for AWS services, as part of your threat model.

**8. Come up with threats**

Your aim here is to try to come up with answers to the question *“What can go wrong?”* There isn’t any canonical list that lists all the possible threats, because determining threats depends on the context of the workload feature that’s under assessment, and the types of threats that are unique to a given industry, geographical area, and so on.

Coming up with threats requires brainstorming. The brainstorming exercise can be facilitated by using a mnemonic like [STRIDE](https://en.wikipedia.org/wiki/STRIDE_(security)) (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege), or by looking through threat lists like the [OWASP Top 10](https://owasp.org/www-project-top-ten/) or [HiTrust Threat Catalog](https://hitrustalliance.net/product-tool/hitrust-threat-catalogue/) to get the ideas flowing.

Through this process, it’s recommended that you develop and contribute to a threat catalog that is contextual to your organization and will accelerate the brainstorming process going forward, as well as drive consistency in the granularity of threats that you model.

**9. Identify and evaluate mitigations**

Here, your aim is to identify the mitigations (security controls) within the workload design and evaluate whether threats have been adequately addressed. Keep in mind that there are typically multiple layers of controls and multiple responsibilities at play.

For your own in-house applications and code, you would want to review the mitigations you’ve included in your design—including, but not limited to, input validation, authentication, session handling, and bounds handling.

Consider all other components of your workload (for example, software as a service (SaaS), infrastructure supporting your [COTS](https://en.wikipedia.org/wiki/Commercial_off-the-shelf) applications, or components hosted within your on-premises data centers) and determine the security controls that are part of the workload design.

When you use AWS services, Security and Compliance is a shared responsibility between AWS and you as our customer. This is described on the [AWS Shared Responsibility Model](https://aws.amazon.com/compliance/shared-responsibility-model/) page.

This means, for the portions of the AWS services that you’re using that are the responsibility of AWS (Security *of* the Cloud), the security controls are managed by AWS, along with threat identification and mitigation. The distribution of responsibility between AWS (Security *of* the Cloud) and you (Security *in* the Cloud) depends on which AWS service you use. Below, I provide examples of *infrastructure*, *container*, and *abstracted* AWS services to show how your responsibility for identifying and mitigating threats can vary:

* [Amazon EC2](https://aws.amazon.com/ec2/) is a good example of an *infrastructure* service, where you are able to access a virtual server in the cloud, you get to choose the operating system, and you have control of the service and all aspects you run on it—so you would be responsible for mitigating the identified threats.
* [Amazon Relational Database Service (Amazon RDS)](http://aws.amazon.com/rds) is a representative example of a *container* service, where there is no operating system exposed for you, and instead AWS exposes the selected database engine to you (for example, MySQL). AWS is responsible for the security of the operating system in this example, and you don’t need to devise mitigations. However, the database engine is under your control as well as all aspects above it, so you would need to consider mitigations for these areas. Here, AWS is taking on a larger portion of the responsibility compared to infrastructure services.
* [Amazon S3](https://aws.amazon.com/s3/), [AWS Key Management Service (AWS KMS)](https://aws.amazon.com/kms/), and [Amazon DynamoDB](https://aws.amazon.com/dynamodb/) are examples of an *abstracted* service where AWS exposes the entire service control plane and data plane to you through the service API. Again, here there are no operating systems, database engines, or platforms exposed to you—these are an AWS responsibility. However, the API actions and associated policies are under your control and so are all aspects above the API level, so you should be considering mitigations for these. For this type of service, AWS takes a larger portion of responsibility compared to container and infrastructure types of services.

While these examples do not encompass all types of AWS services that may be in your workload, they demonstrate how your Security and Compliance responsibilities under the Shared Responsibility Model will vary in this context. Understanding the balance of responsibilities between AWS and yourself for the types of AWS services in your workload helps you scope your threat modeling exercise to the mitigations that are under your control, which are typically a combination of AWS service configuration options and your own workload-specific mitigations. For the AWS portion of the responsibility, you will find that [AWS services are in-scope of many compliance programs](https://aws.amazon.com/compliance/services-in-scope/), and the audit reports are available for download for AWS customers (at no cost) from [AWS Artifact](https://aws.amazon.com/artifact/).

Regardless of which AWS services you’re using, there’s always an element of customer responsibility, and this should be included in your workload threat model.

Specifically, for security control mitigations for the AWS services themselves, you’d want to consider security controls across domains, including these domains: Identity and Access Management (Authentication/Authorization), Data Protection (At-Rest, In-Transit), Infrastructure Security, and Logging and Monitoring. AWS services each have a [dedicated security chapter](https://docs.aws.amazon.com/security/) in the documentation, which provides guidance on the security controls to consider as mitigations. When capturing these security controls and mitigations in your threat model, you should aim to include references to the actual code, IAM policies, and [AWS CloudFormation](http://aws.amazon.com/cloudformation) templates located in the workload’s infrastructure-as-code repository, and so on. This helps the reviewer or approver of your threat model to get an unambiguous view of the intended mitigation.

As in the case for threat identification, there’s no canonical list enumerating all the possible security controls. Through the process described here, you should consciously develop baseline security controls that align to your organization’s control objectives, and where possible, implement these baseline security controls as platform-level controls, including AWS service-level configurations (for example, encryption at rest) or guardrails (for example, through [service control policies](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_scps.html)). By doing this, you can drive consistency and scale, so that these implemented baseline security controls are automatically inherited and enforced for other workload features that you design and deploy.

When you come up with the baseline security controls, it’s important to note that the context of a given workload feature isn’t known. Therefore, it’s advisable to consider these controls as a negotiable baseline that you can deviate from, provided that when you perform the workload threat modeling exercise, you find that the threat that the baseline control was designed to mitigate isn’t applicable, or there are other mitigations or compensating controls that adequately mitigate the threat. Compensating controls and mitigating factors could include: reduced data asset classification, non-human access, or ephemeral data/workload.

To learn more about how to start thinking about baseline security controls as part of your overall cloud security governance, have a look at the [How to think about cloud security governance](https://aws.amazon.com/blogs/security/how-to-think-about-cloud-security-governance/) blog post.

**10. Decide when enough is enough**

There’s no perfect answer to this question. However, it’s important to have a risk-based perspective on the threat modeling process to create a balanced approach, so that the likelihood and impact of a risk are appropriately considered. Over-emphasis on “let’s build and ship it” could lead to significant costs and delays later. Conversely, over-emphasis on “let’s mitigate every conceivable threat” could lead to the workload feature shipping late (or never), and your customers might move on. In the recommendation I made earlier in the “Assemble the right team” section, the selection of personas is deliberate to make sure that there’s a natural tension between shipping the feature, and mitigating threats. Embrace this healthy tension.

**11. Don’t let paralysis stop you before you start**

Earlier in the “Break the workload down into smaller parts” section, I gave the recommendation that you should scope your threat models down to a workload feature. You may be thinking to yourself, “We’ve already shipped <X number> of features, how do we threat model those?” This is a completely reasonable question.

My view is that rather than go back to threat model features that are already live, aim to threat model any new features that you are working on now and improve the security properties of the code you ship next, and for each feature you ship after that. During this process you, your team, and your organization will learn—not just about threat modeling—but how to communicate effectively with one another. Make adjustments, iterate, improve. Sometime in the future, when you’re routinely providing high quality, consistent and reusable threat models for your new features, you can start putting activities to perform threat modeling for existing features into your [backlog](https://www.scrum.org/resources/what-is-a-sprint-backlog).

**Conclusion**

Threat modeling is an investment—in my view, it’s a good one, because finding and mitigating threats in the design phase of your workload feature can reduce the relative cost of mitigation, compared to finding the threats later. Consistently implementing threat modeling will likely also improve your security posture over time.

I’ve shared my observations and tips for practical ways to incorporate threat modeling into your organization, which center around communication, collaboration, and human-led expertise to find and address threats that your end customer expects. Armed with these tips, I encourage you to look across the workload features you’re working on now (or have in your backlog) and decide which ones will be the first you’ll threat model.

If you have feedback about this post, submit comments in the Comments section below.

Want more AWS Security how-to content, news, and feature announcements? Follow us on [Twitter](https://twitter.com/AWSsecurityinfo).