Identify Attacks and Device Countermeasures

The Threat Modeling Playbook

To select the right security measures requires a precise understanding of potential attacks. Threat modeling identifies weaknesses early and creates transparency around potential attack vectors before critical flaws ever reach production. Learn how to implement this deliberate, strategic discipline in your DevOps practices to spot weaknesses much earlier within the delivery lifecycle.

How to identify threats and derive targeted countermeasures

An Introduction to Threat Modeling

by Clemens Hübner

IT systems are constantly exposed to new threats, and mitigating these threats is anything but trivial. Selecting the right security measures requires a precise understanding of potential attacks. The threat modeling method enables a systematic, proactive approach to security. This article explains how it can be used to detect threats early on, what steps are important for this, and why it even helps Santa Claus.

Santa Claus has a security problem. Gifts keep disappearing from the vault. Children report receiving the wrong presents. And his cheeky elves can no longer be trusted.

But what can be done? Add more locks to the safe? Monitor the packaging more closely? Double and triple check the wish lists? Santa Claus can't take care of everything at once—and Christmas is fast approaching!

This scenario is all too familiar in the IT world: ever-changing threats necessitate the introduction of new security measures, but choosing which ones to implement is anything but trivial. Which countermeasures really help? When developing new software systems, developers and architects often find themselves faced with an overwhelming number of potential risks.

The desire for tailored, prioritized measures requires a precise understanding of the threats. After all, the question is not whether a system will be attacked, but when and how.

Threat modeling transforms security from a downstream, reactive discipline into a proactive strategy. It offers a systematic and targeted approach to identifying and mitigating security threats early on in the development cycle. Instead of fixing bugs in an uncoordinated manner or blindly implementing costly security measures, threat modeling creates the necessary foundation for the strategic integration of security controls. As an established method for considering security aspects, threat modeling also facilitates the exchange of knowledge within a development team.

This article aims to introduce the central ideas of threat modeling and provide a standard approach for getting started with the method. But beware: there is no one right way to do threat modeling. Variations and adaptations may be appropriate depending on the organization, prior knowledge, or applications. The following articles will take a closer look at individual aspects of this.

The 4-Question Framework

Security should not be viewed as an optional add-on, but rather as an architectural decision. To facilitate this systematic decision-making process, established threat modeling processes are usually based on the 4-question framework. This framework, developed by Adam Shostack, serves as an anchor for the approach and divides the threat analysis process into four clear, sequential steps:

1. What are we building?
2. What can go wrong?
3. What do we do about it?
4. Did we do a good job?

These four questions force development teams and architects to view the system from an attacker's perspective and take a problem-solving approach. They ensure that all relevant aspects—from system architecture and potential vulnerabilities to the validation of countermeasures—can be taken into account.

Santa Claus now wants to systematically explore the threats to his gift logistics. In the following sections, we will demonstrate the application of the 4-Question Framework using the case study of his gift wrapping machine.

Step 1: Understanding the system – what are we building?

Santa Claus thinks: Actually, it's not just the safe he has to protect. After all, the presents have to be wrapped first. He has a machine that reads the wishes from the list and wraps the corresponding gifts. What else is involved? And which elves actually have access?

Threat modeling typically starts with the question: What exactly do we want to protect? To enable systematic threat analysis, it is advisable to define the boundaries, components, interactions, and critical assets of the system early on in the architecture modeling process.

The modeling and visualization of the system under consideration can be done in various ways. From a security perspective, the flow, processing, and storage of data are often of particular interest. Therefore, modeling as data flow diagrams (DFDs) is a good option. The DFD (Fig. 1) is a powerful, easy-to-understand tool that depicts the flow of information through the system and identifies the basic elements (processes, data flows, data storage, external entities). Finally, trust boundaries can be defined when creating the model – these are the lines where security requirements change. Whether DFDs, sequence diagrams, or other architectural approaches are chosen, the goal remains to create a clear, preferably visual basis for further, continuous analysis.

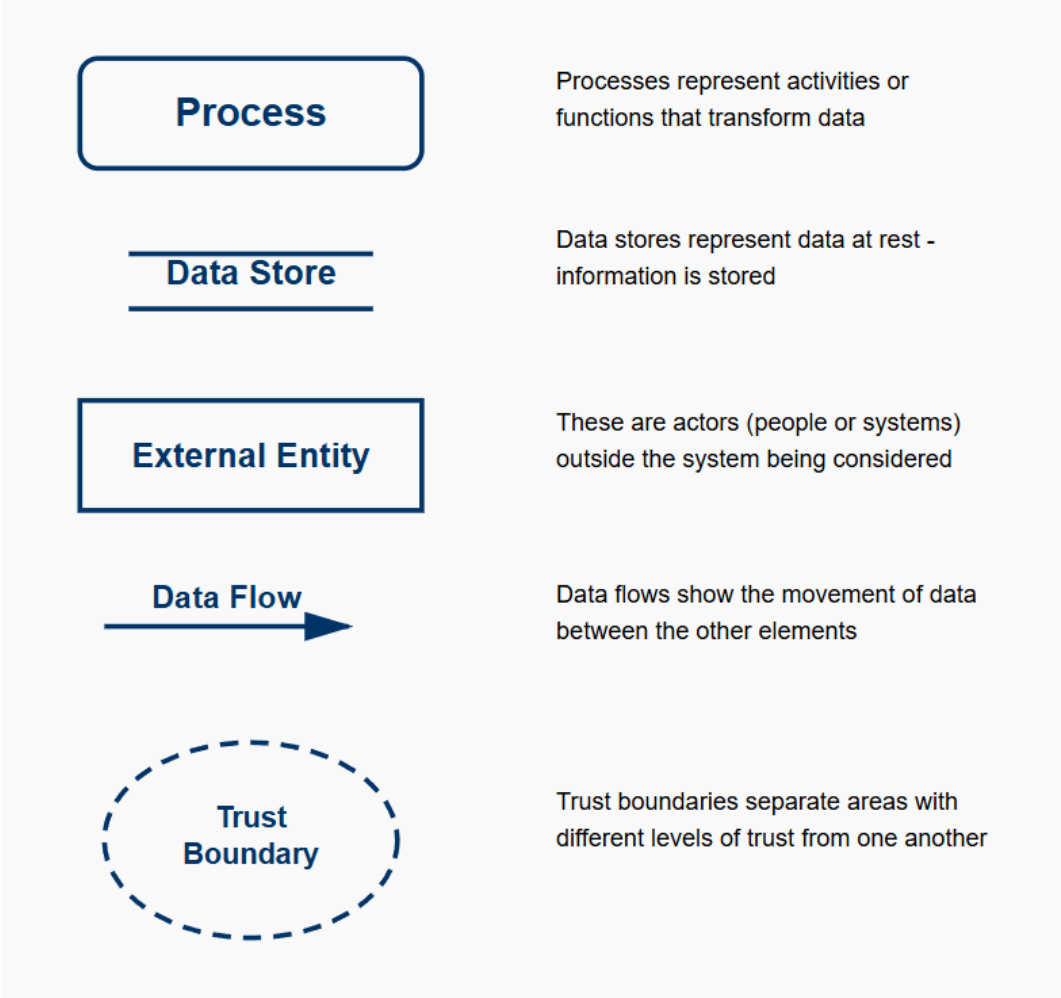

FIGURE 1 The elements of a data flow diagram (DFD)

In the case of the gift wrapping machine, the following components can be identified (Fig. 2):

1. Wish lists: the data storage device on which customers' wishes are recorded
2. Operator terminal: the interface through which Santa Claus or authorized elves control the machine
3. Selection and packaging machine: the central process that physically assembles and packages the gifts
4. Gift vault: the secure data storage device for the finished packaged goods

Identifying these system components leads directly to the definition of critical assets. Assets are not only physical components; above all, they are the information or capabilities of the system that are worth protecting.

Here, it helps to define directly for each asset which security objectives (confidentiality, integrity, availability) must be prioritized and what the violation of this objective means in practical terms. In the case of the gift wrapping machine, we focus on two main assets: The wish list is essential for the integrity of the entire process, as its manipulation would lead to incorrectly assigned gifts. The gift vault, on the other hand, primarily requires secure availability so that the gifts can be retrieved on time for delivery.

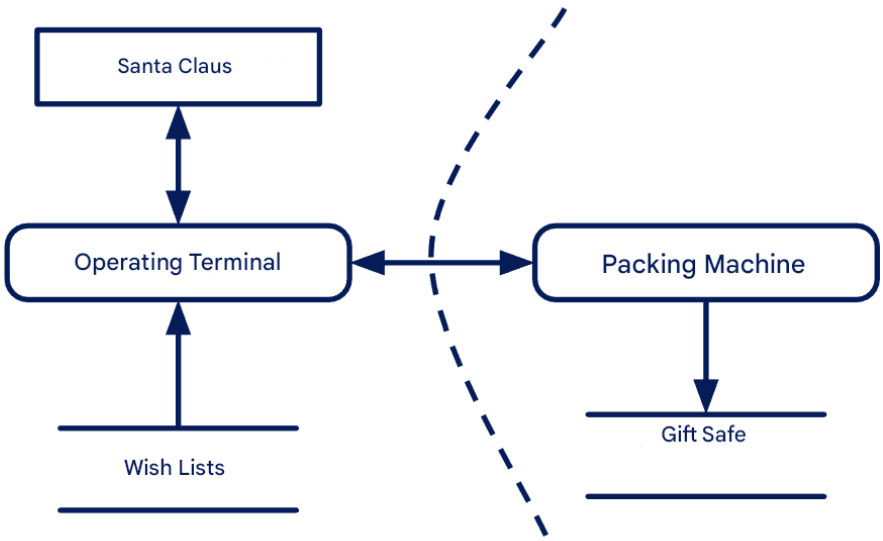

FIGURE 2 A minimal data flow diagram for Santa's gift machine

With this understanding of the system, components, and the assets it contains, we can now move on to the second step.

Step 2: Identify threats – what could go wrong?

Santa Claus has understood the components of his system. But what are the threats? And how can he ensure that he has thought of everything?

In this step, we turn to the critical question: What can go wrong? Answering this question begins with considering the threat actors. The spectrum of attackers ranges from script kiddies, who act with little knowledge and no specific motivation, to opportunistic criminals, terrorists, and state actors. Ethical hackers, competitors, and insiders with their respective motivations are also potential attackers.

In the Santa Claus scenario, the cheeky elves are the most significant threat actors as "insiders." They have access to the otherwise isolated system, which no one outside the North Pole knows about anyway.

Modeling the attacker is helpful in order to subsequently identify appropriate threats. Different attacks are relevant depending on knowledge, resources, motivation, and capabilities.

Systematic analysis using STRIDE

To identify the wide range of potential threats in a structured manner, it is advisable to use a framework. STRIDE is a popular and well-established option. Alternatives and additions that follow a different systematic approach are also conceivable.

The name STRIDE is an acronym, with each of the six letters standing for a class of attack: Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. Each area is directly linked to fundamental security objectives, ensuring a holistic approach.

Let‘s take a look at the six classes of STRIDE.

Identification of threats according to STRIDE

Spoofing

This threat affects the authenticity and ability of an attacker to impersonate another entity (person or system). A successful spoofing attack undermines the system‘s identity controls. In the context of the gift wrapping machine, an elf could attempt to forge or steal Santa‘s access card at the operator terminal in order to impersonate the boss and take over privileged machine controls.

Tampering

This is a core threat that attacks the integrity of the system. It occurs when unauthorized changes are made to data, configurations, or processes. The elves could manipulate the wish list stored in the data memory by changing entries, for example, to replace high-value gifts with inferior ones. This directly leads to a breach of the critical asset’s integrity.

Repudiation

This threat is related to the non-repudiation of actions. Repudiation occurs when an entity can deny having performed an action because there is no audit-proof evidence or audit logs. In the case of the elf machine, this problem becomes acute when the manual shift log is used: if an elf discovers sabotage on the machine, they can deny it because the shift log can be manipulated, and there are no tamper-proof logs to prove guilt.

Information Disclosure

This threat targets the confidentiality of the system and means that sensitive data is disclosed to unauthorized parties. In this scenario, the packaging itself could reveal confidential information. If the elves can draw conclusions about the valuable contents of the gift based on its external shape, size, or special labels, confidentiality is compromised, even though the data has not formally been stolen from the database.

Denial of Service

This is another major threat that primarily affects system availability and prevents authorized users from using the service or asset when they need it. Elves could block access to the gift vault by triggering access control disruptions. The critical asset, the gift vault, is no longer available, which disrupts the continuity of delivery.

Elevation of Privilege

This threat occurs when an attacker exploits a vulnerability to gain higher privileges than they are entitled to. The packaging machine has a secret monitoring hatch that is reserved for Santa Claus. If elves learn about this feature and exploit a vulnerability to gain unauthorized access to this privileged feature, they gain extended system control, which constitutes an elevation of privileges.

If STRIDE is applied consistently to all components of the data flow diagram, you will obtain a comprehensive list of specific threat scenarios. This threat list forms the necessary basis for the next, third step.

Step 3: Find countermeasures – what can we do about it?

Santa Claus is thrilled: now he has a very good overview of the threats. He hadn‘t even thought of some of them before. But so far, he‘s not any safer. What does he need to do now?

The strength of threat modeling lies in its consistency: once the threats have been described in detail and understood, the necessary protective measures (security controls) can be derived directly and evaluated. Ultimately, the aim is to mitigate threats, not just document them. The architecture of the gift wrapping machine must now be adapted in such a way that the cheeky elves‘ attempts to attack it become wasteful or impossible.

Derivation of security controls

The systematic nature of the STRIDE model ensures that each threat generates a specific security requirement. The implementation of these controls translates the threat models into concrete work packages for developers and architects.

- Spoofing attacks authenticity and requires controls that ensure robust identity verification. It is advisable to introduce mechanisms that are not based solely on the possession of a single factor. In the case of a spoofing attack, in which an elf attempts to use Santa's stolen access card at the service terminal, it is necessary to switch to strong authentication. This means integrating multi-factor authentication (MFA), which requires the entry of a one-time code or the use of a biometric feature in addition to the card.
- Tampering targets the integrity of data and processes. To prevent this, measures must be taken to ensure that data remains unalterable and verifiable. Integrity checks are essential for critical wish lists. To this end, the list is secured with cryptographic hashes or digital signatures. Every processing operation initiated by the machine for selection and packaging begins with a hash check to ensure that the data has not been altered since it was created.
- Repudiation undermines the non-repudiability of actions. Controls to mitigate this threat focus on tamper-proof and complete logging. To prevent repudiation, the vulnerable manual shift log must be replaced with audit-proof audit logs. These logs must securely and tamper-proof store who did what on the machine and when. This allows the responsibility of the elves for acts of sabotage to be systematically proven retrospectively.
- Information disclosure compromises confidentiality and requires measures to protect data at rest and during processing. This also applies to indirectly disclosed information. Confidentiality controls are necessary to combat information disclosure through the packaging itself. This includes the use of neutral, uniform packaging and the concealment of all identifying features so that even if the elves come into physical contact with the packages, they cannot draw any conclusions about their valuable contents.
- Denial of service attacks target the availability of the system and must be mitigated by controls that ensure redundancy, resilience, and capacity. To prevent denial of service at the gift vault and ensure the required availability, availability and resilience measures must be strengthened. This includes protecting against overload of the access system and ensuring that critical paths for removing gifts function redundantly and stably, even if individual Secret Santas attempt to block components.

- Elevation of privilege is mitigated by strong authorization and access control mechanisms. As a basic strategic requirement, the principle of least privilege must be strictly applied. To prevent unauthorized access to the secret surveillance hatch, this means that the privileges of the elves are consistently limited to their respective tasks, so that even if a vulnerability is exploited, it is very difficult to gain higher system control.

This illustrative derivation is intended to show how concrete threats can lead to concrete countermeasures. However, it is not always possible or desirable to implement every measure directly.

Step 4: Retrospective – did we do a good job?

Santa Claus is satisfied: he now has a good understanding of the threat he faces, i.e. what the elves might be plotting and how he can protect himself against it. This will get him through this Christmas season – but what happens after that?

The fourth step of the 4-Question Framework – Did we do a good job? – is often neglected in practice. However, this review is essential to ensure that threat modeling does not remain a one-time activity without lasting results. This validation step transforms threat modeling from a documentation process into a proactive strategy for continuous security improvement. Only with this step can you determine whether the modeling was worth the time – and also how to proceed.

Possible questions that can help with this are:

- Has every identified threat (e.g., every STRIDE entry) been addressed by at least one security control? Are there still threats that have been deliberately defined as an accepted residual risk?
- When creating the data flow diagram (DFD), did we correctly identify all critical data flows, external entities, and trust boundaries? Are there any critical assets missing that are now left unprotected?
- Were the identified threat actors and their motivations (in the example, the cheeky elves) modeled realistically? Would we need to adjust the model if the attacker‘s resources or capabilities changed?
  - Do the derived controls lead to an economic reduction in risk, or do the costs of the controls exceed the potential damage?
- What steps do we need to take for further implementation?
  - When should we perform threat modeling again?

Continuous threat modeling

Threat modeling should never be viewed as a one-time, completed activity, as systems, attackers, and the resources available to them change over time. The question of “What now?” leads directly to the integration of the method into the development cycle (SDLC).

The following tips will help you to anchor the results of threat modeling in the long term:

1. Iterative revalidation: The system will continue to evolve architecturally. New components, new external interfaces, or changed interactions require renewed modeling and threat analysis. Threat modeling should be integrated into every sprint planning or major release as a continuous process.
2. Link to risk management: Not all identified threats or required controls can be implemented immediately. The results must be transferred to an overarching risk management system in order to decide which residual risks must be accepted or prioritized for mitigation.

3. Knowledge transfer and documentation: The findings from threat modeling (DFDs, STRIDE results, security controls) must remain accessible and understandable to the development team. This helps to share knowledge within the development team and embed security as an architectural decision.
4. Various methods: Different methods, some of which are presented here in this issue, not only enable varied sessions. They also create different perspectives on different aspects of a system‘s security.
5. Get started: Even initial, small sessions help to familiarize yourself with the method and find initial results. Any threat model is better than no threat model!

Conclusion

Threat modeling shifts security from a reactive scramble to a deliberate, strategic discipline. By clearly understanding the system, identifying realistic threats, and deriving targeted countermeasures, teams can make informed architectural decisions rather than relying on guesswork. The Santa Claus example shows how even seemingly whimsical scenarios reveal the power of a structured approach. Ultimately, integrating threat modeling into regular development cycles ensures that security remains continuous, scalable, and truly effective.

Clemens Hübner has been working as a software security engineer at inovex since 2018. There, he supports development projects at the design and implementation stages, provides consulting on DevSecOps, and conducts training sessions. As a speaker, he shares his expertise at national and international conferences.

From Misconfigurations to Insider Threats: A Threat Analysis

Threat Modeling for Infrastructure as Code

by Larysa Bondar

Infrastructure as Code (IaC) frameworks are powerful tools, but they come with their own security risks. Misconfigurations, insecure defaults, or insufficient access controls can be exploited by attackers. Threat modeling helps identify weaknesses early and creates transparency around potential attack vectors before critical flaws ever reach production. This article introduces practical methods and automation approaches, showing how continuous threat modeling at the infrastructure level can firmly embed security into DevOps practices.

When building a house, you carefully review the blueprint before laying the first brick. No one wants to discover later that the foundation is unstable or that a door opens into a wall. Infrastructure as Code works the same way: even a small misconfiguration—an open port here, overly permissive database access there—can later become a backdoor for attackers.

Threat modeling acts as the security blueprint. It reveals where weaknesses might exist: Where could an intruder get in? Is the door secure enough? Are there open windows? It creates transparency around potential risks before the "house"—the cloud infrastructure—is even built. This results in stable, secure environments where security is not added afterward but forms part of the foundation from the very beginning.

Security Challenges Specific to Infrastructure as Code

While traditional software development usually focuses on user interactions and application logic, Infrastructure as Code describes the entire technical foundation—networks, access permissions, and cloud services. A single line of code can suddenly determine whether a database is publicly accessible or properly secured.

Misconfigurations no longer spread gradually. With a single terraform apply or kubectl apply, they can immediately affect dozens or even hundreds of systems. What starts as a minor oversight can quickly turn into a widespread security vulnerability.

Another challenge is hidden dependencies between resources: IaC templates link many services and configurations—often implicitly and in ways that are difficult to trace. Even a small change to a security group rule can unintentionally expose entire service chains. These side effects overwhelm traditional code reviews and complicate threat modeling.

Infrastructure as Code introduces specific security challenges that traditional threat models often fail to fully cover [1]: API keys, access tokens, or passwords frequently end up—unintentionally and in plain text—in IaC repositories, module files, or CI/CD pipelines. If these secrets are reused across multiple systems or paired with overly permissive Identity and Access Management (IAM) policies, attackers can move laterally and, in the worst case, take over entire system landscapes.

This becomes especially critical when a module with excessive permissions is reused multiple times, multiplying the attack surface. This is a phenomenon known as secret sprawl.

Manual changes made to infrastructure after automated IaC deployment create discrepancies between the declarative IaC state and the actual runtime state—a phenomenon known as configuration drift. This opens new security gaps that cannot be detected by analyzing IaC templates alone. Using public third-party IaC modules or templates introduces further risks, as outdated or insecure defaults may silently enter production environments.

Many IaC templates rely on insecure default configurations for convenience, leading to systematic vulnerabilities when reused. Some IaC tools even store sensitive information such as resource IDs or configuration details in state files. I'll leave it to you to imagine what happens if those files are exposed.

Identifying these threats is only the first step. The real challenge lies in addressing them systematically and designing infrastructure that is secure, transparent, and auditable from the outset. This is especially true in modern DevOps pipelines, where continuous delivery enables rapid, automated infrastructure changes. While this boosts productivity for development teams, it can be a nightmare for security teams. The more frequent the rollouts, the harder it becomes to perform timely threat analysis. One-off security workshops are no longer sufficient in this dynamic environment [2].

These three dimensions—scale, dependencies, and speed—require a rethink of IaC threat modeling. Instead of being an occasional workshop activity, threat modeling must become a fixed component of the DevOps lifecycle. Only then can threats be identified directly within IaC templates. Tools and automation help make attack vectors visible, prioritize risks, and detect issues already during design.

Shift-Left Security in Infrastructure as Code

Infrastructure as Code is a central element of modern DevOps processes. However, security is often not deeply embedded and is instead addressed late in the development cycle. DevSecOps extends DevOps by integrating security into processes, systems, and infrastructure from the very beginning. Security is "shifted left": tests, security checks, and safeguards are applied as early as possible in the Software Development Lifecycle (SDLC).

This establishes security not as an obstacle but as an integral part of the DevOps workflow, with automation, clear processes, and shared responsibility.

When security checks occur only at the end of a pipeline, vulnerabilities are often discovered in already deployed environments, increasing remediation costs or allowing attackers to exploit them first. The shift-left approach reverses this: instead of costly post-deployment fixes, security is considered while writing Terraform code, Ansible playbooks, or Kubernetes manifests. Developers verify policy compliance while creating templates, making security part of the code itself.

DevSecOps is primarily a cultural shift. Security becomes a team responsibility. Developers take on more accountability, and security teams move closer to day-to-day development. Shared workshops and training sessions spread knowledge and best practices across teams.

In IaC environments, shift-left security is not optional; it is essential. Misconfigurations, hidden dependencies, and the speed of continuous delivery demand that security be embedded from the start.

Continuous Threat Modeling and Threat Modeling as Code

In traditional software development, threat modeling often happens once, in a workshop at the start of a project. That used to be reasonable. But with agile methods, microservices architectures, and continuous delivery pipelines now standard, this approach no longer works. With every commit, new container image, and updated Kubernetes configuration, the attack surface changes.

This makes continuous threat modeling necessary. Security models must evolve alongside the code. That means integrating threat analysis directly into CI/CD pipelines. With every commit, not only do unit and integration tests run, but they also check against defined security models [2].

Manual threat modeling is difficult. High-quality threat models require experienced specialists. Systems constantly grow and change. Every technology and dependency introduces new vulnerabilities. Maintaining consistency across tools, frameworks, and documentation formats is challenging.

This is where automation comes in. Software can dynamically represent systems, analyze dependencies, and evaluate rules. Just as infrastructure is defined using IaC, threat scenarios, assumptions, and risks can be represented in JSON or YAML, versioned, and validated [3]. Instead of manually updating diagrams, the model lives directly in the repository. Every change can be automatically validated, making threat modeling reproducible and scalable [2].

Two approaches are commonly distinguished: Threat Modeling from Code and Threat Modeling with Code:

- **Threat Modeling from Code** analyzes source code, configurations, or infrastructure definitions to derive a threat model. Tools compare these inputs against known risks or rules and generate reports identifying potential vulnerabilities. Code itself becomes the input for threat identification [4].
- **Threat Modeling with Code** describes existing systems—entities, data flows, dependencies, or event sequences—in code form. Developers model systems using programming languages or Architecture Description Languages (ADLs) such as AADL or Acme. These models are then interpreted to automatically identify threats [4].

In both approaches, output quality depends heavily on input quality: garbage in, garbage out. With clean input data and valid rules, two major pain points disappear: reliance on highly specialized expertise and manual maintenance of system models.

Combining these approaches with modern IaC and DevOps practices leads to Threat Modeling as Code. Threat models are described in machine-readable formats like YAML or JSON, versioned as source code, and automatically derived from existing code. They integrate directly into CI/CD pipelines, making threat modeling continuous, automated, and collaborative. Threat models are code [2].

Continuous threat modeling is a logical progression from threat modeling as code. This modern approach is based on a few key principles [4]:

1. **Team knowledge beats external knowledge**: no one understands a system better than the product team.
2. **Agile, accessible, and practical**: threat modeling must adapt to real workflows.
3. **Real-time learning curve**: quality analysis improves as teams gain experience (“increasing returns learning curve”).
4. **Model and reality must match**: threat models must always reflect the current system state.
5. **Continuous improvement**: today’s model should be better than yesterday’s.
6. **Actionable value**: insights must help identify and mitigate real risks.

With these principles, threat modeling becomes a continuous, team-driven process that embeds IaC security from the start and strengthens infrastructure resilience sustainably.

Common Threat-Modeling Methods for IaC

Not all threat-modeling methods are equally suitable for Infrastructure as Code. In practice, two approaches have proven effective: **STRIDE** and **DREAD**.

STRIDE stands for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. It systematically identifies potential attack vectors and translates well to infrastructure components such as storage, networking, and access management.

DREAD is a scoring-based framework evaluating Damage Potential, Reproducibility, Exploitability, Affected Users, and Discoverability. IaC environments benefit especially from DREAD for prioritization: scores can be automated, visualized in dashboards, and linked to SLAs or runbooks.

In practice, IaC threat modeling often combines both: STRIDE provides structured qualitative analysis, while DREAD turns findings into prioritized, operational measures. Small teams often start with pragmatic STRIDE workshops at the template or module level. Larger organizations complement this with DREAD scoring and automated CI/CD evaluations.

Example: Threat Modeling with IaC

To illustrate the approach, consider a simple example. Our system architecture is deployed in Azure and defined using Terraform, but the methodology applies to any cloud or IaC tool.

The architecture includes:

- **Resource Group**: Container for all resources
- **Azure AD**: Identity management and role-based access control
- **Azure DNS**: Name resolution in the network
- **Virtual Network (VNet)**: Virtual network with two subnets
- **App Subnet**: Contains the application VM
- **DB Subnet**: Contains the database
- **Network Security Groups (NSG)**: Network firewalls for app and DB subnets
- **Azure Load Balancer**: Distribution of HTTPS traffic
- **Virtual Machine (VM):** Application component that accesses the DB via HTTPS
- **Azure Database (e.g., PostgreSQL)**: Persistent storage of application data
- **Managed Identity**: Ensures access to the database via Azure AD
- **DNS zones and records**: For name resolution between components

A generalized architecture is illustrated in **Figure 1**.

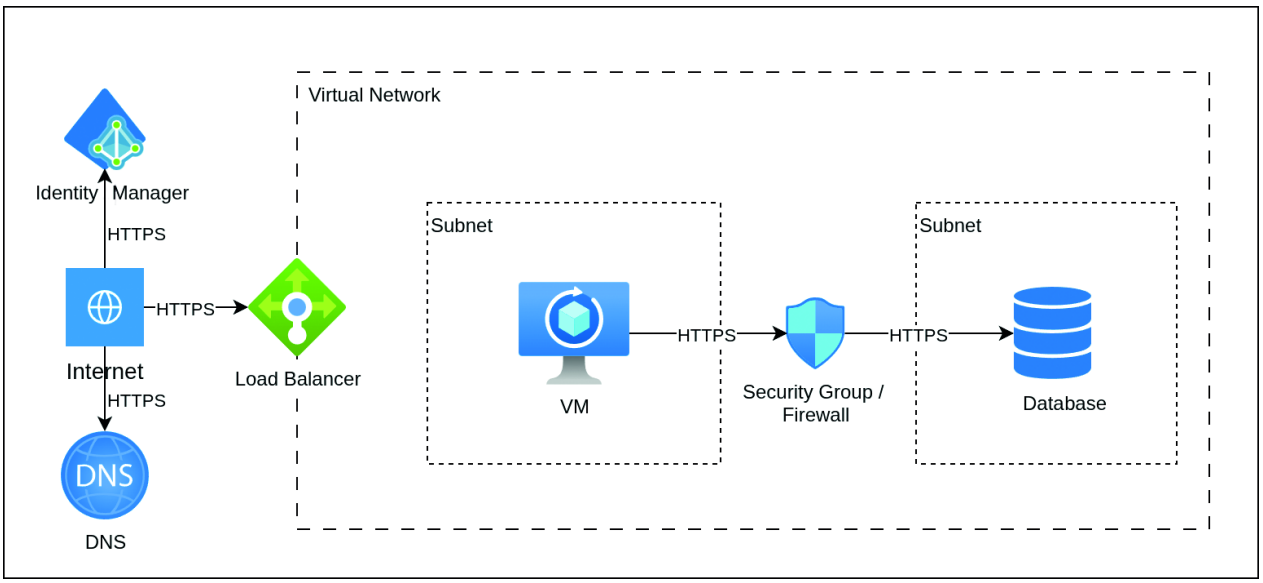

FIGURE 1 Generalized example architecture

We will now look at our Terraform configuration, define trust boundaries, and apply STRIDE and DREAD methods to them. The data flow diagram is omitted in this example because we already know how it works, and the process is exactly the same for IaC. Listing 1 shows an example of what this might look like. Of course, there are several trust boundaries and vulnerabilities, but if we want to describe them all, we'll be busy all day.

Listing 1

```
VNet-Subnets und Network Security Groups
resource "azurerm_subnet" "app_subnet" {
  ...
}
resource "azurerm_subnet" "db_subnet" {
  ...
}
resource "azurerm_network_security_group" "app_nsg" {
  security_rule {
    name                       = "HTTPS"
    source_address_prefix      = "*"
  }
}
resource "azurerm_network_security_group" "db_nsg" {
  security_rule {
    name                       = "Internal-HTTPS"
    source_address_prefix      = azurerm_subnet.app_subnet.address_prefixes[0]
  }
}
```

STRIDE

- **Spoofing**: External access to the app subnet enables identity spoofing
- **Tampering**: Overly permissive NSG rules allow packet manipulation
- **Information Disclosure**: Open NSGs increase data exposure risk

DREAD:

- Public SSH access to the VM (NSG rule with source_address_prefix = "*", port 22)
  - Damage Potential: 8
- Successful SSH access can mean root privileges on the VM and lead to complete system takeover
  - Reproducibility: 10
- The open SSH port is easy to find via port scanning; attacks can be repeated as often as desired
  - Exploitability: 9
- SSH is widely used, and automated exploits and brute force tools exist to exploit unsecured SSH access
  - Affected Users: 10

- All users and services running on the VM would be potentially affected
  - Discoverability: 9
- Port 22 is the standard port for SSH and, therefore, easy to discover
  - Score: 9.2 (high)

Of course, it's a lot of fun to do something like this manually. Unfortunately, life is too short, so smart people have made the process much faster, easier, and less error-prone with automation tools.

Tools and Best Practices

There are many open-source and enterprise tools that simplify IaC threat modeling. Here are a few examples:

- **threatcl**: a configuration language for threat modeling workflows [5]
- **TerraScan**: open-source tool for static code analysis of IaC, checks Terraform, Kubernetes, and CloudFormation configurations for security vulnerabilities and compliance violations [6]
- **tfsec**: open-source tool for static analysis of Terraform code, detects insecure configurations, dangerous patterns, and missing encryption [7]
- **Checkov**: checks for typical misconfigurations using predefined and user-defined rules; supports Terraform, AWS CloudFormation, Azure Resource Manager, and Kubernetes manifests [8]
- **kube-review and kubeval**: Tools specifically for Kubernetes that scan manifests for policy compliance and structural weaknesses, well-suited for microservices and container environments [9], [10]
- **OWASP Threat Dragon**: A comprehensive threat modeling tool that is not only suitable for IaC, but is also generally useful in DevSecOps processes [11]

Often, several of these tools are used in parallel to cover different aspects such as cloud specifications, containers, and rights management. However, before we get started and integrate automated threat modeling into our CI/CD pipeline, we should consider some best practices that will make life much easier for all of us. Techvzero suggests the following framework for integrating threat modeling into IaC workflows [1]:

- Delineate and inventory assets: define clear IaC boundaries, consider module boundaries as trust zones, segment development, test, and production environments, and capture critical assets (cloud accounts, IAM entities, CI/CD infrastructure)

- Identify sensitive data flows: recognize regulated data, intellectual property, and trade secrets in the infrastructure setup and document their flow.
- Apply threat modeling methods: use STRIDE to identify technical risks and DREAD for prioritization. PASTA, which was not mentioned in this article, combines threat analysis with business objectives and is usually used as a supplement.
- Implement mitigation as code: embed security policies in infrastructure templates, use secure, reusable modules, and integrate security scanners (e.g., Checkov, Terrascan) early in the pipeline.
- Security controls throughout the IaC lifecycle: pre-commit hooks and IDE plugins for early detection, CI/CD pipeline quality gates with static analysis and policy validation, as well as post-deployment monitoring, dynamic testing, and compliance monitoring.

Conclusion

Here are a few thoughts on what we, as platform engineers and developers, can do to better secure systems:

- Perform security checks during development, not just during review.
- Involve all stakeholders, from DevOps to development to security. Joint workshops and open communication are crucial.
- Automate checks and policy checks with every code change.
- Don't treat security policies as a one-time task, but review and adjust them regularly.
- Configure the CI/CD pipeline so that unstable or insecure IaC changes automatically block deployment until the risks are addressed.
- Establish clear guidelines, versioning, and regular audits.
  This makes security a continuous process.

And that's it. Threat modeling for Infrastructure as Code does not end with Terraform configuration files or Kubernetes manifests. The topic encompasses many different methods and insights into DevSecOps and can be expanded at any time. Good luck convincing the product owner that threat modeling is important for your project!

Larysa Bondar holds a master's degree in Cloud Applications and Security Engineering and works as a Cloud Platform Engineer at inovex GmbH. Her professional focus is on cloud architecture, security, and DevSecOps. She has a particular interest in Infrastructure as Code as well as other concepts in cloud computing and cloud security.

Links & Literatur

[1] https://techvzero.com/iac-threat-modeling-best-practices-guide/
[2] https://dev.to/vaib/continuous-threat-modeling-and-threat-modeling-as-code-tmasc-43c8
[3] https://www.hashicorp.com/en/resources/shifting-threat-modeling-left-automated-threat-modeling-using-terraform
[4] Tarandach, I.; Coles, M. J.: „Threat Modeling"; O'Reilly Media, 2020
[5] https://threatcl.github.io
[6] https://runterrascan.io
[7] https://aquasecurity.github.io/tfsec/v1.20.0/
[8] https://www.checkov.io
[9] https://github.com/anderseknert/kube-review
[10] https://github.com/instrumenta/kubeval
[11] https://owasp.org/www-project-threat-dragon/

Proactive security with shift-left approaches

Enhancing Public Cloud Security through Threat Modelling Techniques

by Romina Druta

Threat Modelling is one proactive way to address cloud security much earlier in the delivery lifecycle, which can help identify problem areas early. This article will help you understand the basics of Threat Modelling and how to approach the cloud tactically with security at the forefront.

The technology landscape has been profoundly disrupted by the cloud. Cloud adoption introduces new infrastructures like containers and reliance on various cloud providers. It facilitates innovation by enabling rapid feature delivery and optimising investments. Furthermore, the cloud transforms businesses by changing models, enabling scalability, and facilitating new product development.

Cloud solutions have become a top priority for many organisations. For a company like Visma, which operates across Europe and Latin America with 1.9 million customers and a large, diversified technology stack, approximately 90% of their Research and Development (R&D) spend is dedicated to cloud solutions.

However, with the shift to the cloud comes significant security considerations. Cybersecurity is a growing concern globally, with projected costs reaching $10.5 trillion in 2025. If cybercrime were a country, it would rank as the third-largest economy.

Security in the cloud presents unique challenges, notably the shared responsibility model, the need for increased visibility, managing multi-cloud environments, and the high speed of innovation. These factors necessitate a proactive approach to identifying and mitigating potential security issues.

According to Orca State of Cloud Security Report [1], common vulnerabilities highlighted include publicly exposed Virtual Machines (VMs) with open ports, public-facing storage buckets with sensitive data, publicly exposed databases with sensitive data, and publicly accessible Kubernetes API servers. A lack of poor configuration of Identity and Access Management (IAM) policies is also a significant concern.

These vulnerabilities can be exploited in attack scenarios, such as cloud ransomware attacks, which typically involve exploitation and initial access, discovery and spread, extraction and exfiltration, and finally, data encryption and a ransom demand. Common vulnerabilities and misconfigured cloud resources are key enablers for such attacks.

To address these challenges proactively, shift-left Security is crucial. This approach positions the security process earlier in the delivery lifecycle, rather than treating it as an afterthought before deployment. One of the shift-left security methods used by practitioners is Threat Modelling.

What is Threat Modelling?

Threat modelling is a structured process used to review the security of any system. It aims to identify problem areas and determine the risk associated with each area. Essentially, it involves thinking like an attacker.

The core components of threat modelling include:

- Identifying the attack actors.
- Determining the impact of attacks.
- Identifying threats to assets.
- Defining follow-up actions.

Attack actors can range from state-sponsored entities and hacktivists to career cybercriminals and insiders. The impact of an attack is typically measured against the core security principles of **Confidentiality, Integrity, and Availability** (CIA), as well as potential **financial losses**.

Identifying your assets is another crucial step. In a cloud context, assets include the cloud platform itself, secrets, databases, storage, compute resources, users and groups, and networking.

Understanding the Attack Surface and Threat Vectors

To understand how a system could be attacked, it's essential to focus on the **attack surface**. This includes elements like Virtual Machines, credentials, and points susceptible to brute force attacks or Command & Control infrastructure.

Threat vectors are the paths attackers use to exploit the attack surface. A cloud ransomware attack, for example, can involve several stages:

- **Exposure**: Initial access through common vulnerabilities, insecurely configured applications, or misconfigured cloud resources.
- **Exploitation & Access**: Gaining entry.
- **Installation**: Deploying malicious components.
- **Discover & Spread**: Scanning for cloud resources and identifying data.
- **Retrieve Encryption Keys & Deployment Keys**: Obtaining necessary components for the attack.
- **Extract & Exfiltrate**: Encrypting data and potentially exfiltrating it.
- **Ransom demand**: The final step.

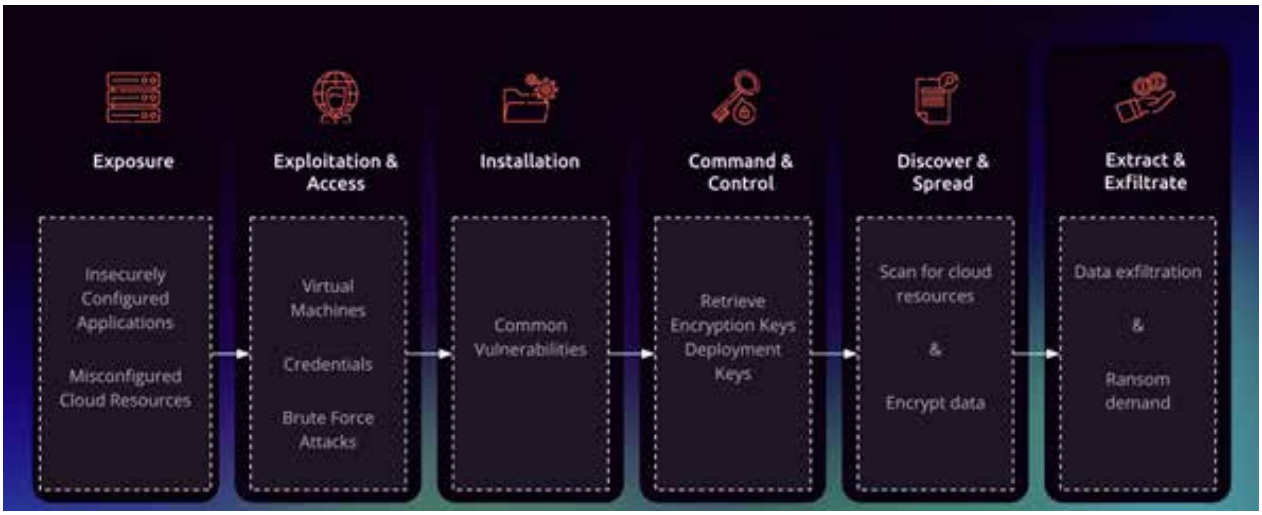

FIGURE 1 Potential threat vectors.

Identifying these threat vectors is key to effective threat modelling.

Approaches to Cloud Threat Modelling

In Visma we thought of an approach that combines automated scanning with structured frameworks.

One approach involves:

- Scanning cloud accounts using a Cloud-Native Application Protection Platform (CNAPP).
- Applying a compliance framework, such as the MITRE ATT&CK framework.
- Grouping assets and identified issues by type and MITRE attack tactic.
- Extracting threat modelling questions based on the results.
  - Using these questions in threat modelling sessions or applying them within the STRIDE-LM framework.

This process can also help improve the security-by-design process.

MITRE ATT&CK Cloud Matrix

The MITRE ATT&CK framework provides a valuable structure for categorising findings by attacker tactics. Applied to the cloud, these tactics describe common adversarial goals and techniques:

- **Initial Access**: How attackers get into the system. This could be through attacking public-facing applications, exploiting misconfigured storage buckets, taking advantage of exposed configurations, public VMs with open ports (like SSH 22 & RDP 3389), publicly exposed K8s Control Plane API or exposed web server configuration.
- **Execution**: How adversaries run malicious code. Examples include exploiting misconfigured cloud functions, implanting malicious containers, or taking over an account with higher privileges.
- **Privilege Escalation**: How adversaries gain higher-level permissions. This can occur if users/roles have more privileges than necessary, Multi-Factor Authentication (MFA) is disabled, Just-in-Time access is missing for privileged users, IAM Users & Roles have administrative privileges.
- **Defense Evasion**: How adversaries avoid detection. Techniques include missing alerts for asset modifications/deletions, disabled audit logs.
- **Credential Access**: How adversaries steal account names and passwords. This might involve network sniffing in the absence of HTTPS enforcement, cloud services brute force attacks, unsecured credentials/misconfigured vaults, users without MFA enabled, cross-account access without MFA, or a root user without MFA.
- **Discovery**: How an adversary gains knowledge about the system and internal network. This can involve querying for domain information, scanning for public assets or network resources, misconfigured network security groups, harvesting data from endpoints (network sniffing), publicly exposed VMs with open ports etc.
- **Lateral Movement**: How adversaries move through a network. This includes using roles that can be exploited by external identities, taking advantage of missing network policies, exploiting exposed environment variables (e.g., AWS Lambda/Azure functions), K8s Namespaces without network policies defined, controllers creating containers with secrets as environment variables, or secrets not encrypted with dedicated keys.
- **Collection**: How adversaries gather data of interest. This involves accessing publicly exposed buckets/databases, taking advantage of unencrypted data, exploiting insecure transfer/unrestricted network access, storage containers publicly exposed.

- **Exfiltration**: How adversaries steal data. This can happen by taking advantage of unrestricted outbound traffic, NSGs with unrestricted outgoing access to the internet, or firewalls bypassed by Cloud Provider Services.
- **Impact**: How adversaries manipulate, interrupt, or destroy systems and data. Examples include taking advantage of overly permissive IAM policies, unprotected assets (secret managers, storages), improper backups, no deletion protection for KeyVaults or storages, no data backup for databases, or no protection in place for resources (lock mechanisms).

STRIDE-LM Framework

Another approach is using the STRIDE-LM framework, which helps structure questions during threat modelling sessions. STRIDE-LM breaks down threats into categories:

Spoofing

Preventing the impersonation of a user or system to gain unauthorised access.

Questions to consider:

- Will IP whitelisting be implemented for publicly exposed resources?
- Do you use firewalls and security groups?
- Do you rate limit and geo-blocking access?
- Will MFA be enabled for all users?
- Will there be users not following the least privilege principle?

Tampering

Preventing unauthorised modification of data or code.

Questions to consider:

- Have you ensured data is encrypted at rest and in transit?
- Will all cloud storage services use TLS 1.2?
- Will you employ File Integrity Monitoring (FIM)?
- Will you have backups in place?
- Will logging and monitoring services be enabled?

Repudiation

Preventing an attacker from denying actions.

Questions to consider:

- Do you have access logs and cloud audit logs, and are they protected/ backed up?
- Do you have Security Log management (SIEM integration)?
- Do you use Digital Signatures?
- Do you regularly audit access and activity logs?

Information Disclosure

Preventing unauthorised parties from accessing confidential information.

Questions to consider:

- Do you classify data and apply stricter controls?
- Do you have Security Groups/Access Control lists?
- Are storage containers denying public access?
- Is data encrypted at rest, and how are keys stored?
- Is network segmentation used?
- Is private networking used for sensitive resources like KeyVaults, Kubernetes clusters, and databases?

Denial of Service (DoS)

Preventing attacks that disrupt or degrade system availability.

Questions to consider:

- Do you use DDoS protection services (e.g., AWS Shield, Google Cloud Armour)?
- Do you have Web Application Firewalls (WAFs)?
- Do you have monitoring to detect DoS attacks early?
- Have you chosen appropriate scalability plans and set scaling limits?
- Have service limits been modified for cloud zones hosting services?

Elevation of Privilege

Preventing attackers from gaining higher access levels than intended.

Questions to consider:

- Do you employ segregation of duties?
- Do you implement least privilege access (e.g., using predefined roles)?
- Do you use Just-in-Time access privilege?
- What RBAC roles are granted?
- Do you use conditional and context-based access control (e.g., checking login region)?
- Do you monitor and log access to cloud resources?

Lateral Movement (LM)

Preventing attackers from expanding control beyond the initial compromise point.

Questions to consider:

- How will you secure and monitor remote access?
- Have you considered jump servers and bastions?
- Will network segmentation and micro-segmentation be in place?

Handling Threat Modelling Findings

Once threats are identified through these processes, the findings are addressed. This involves:

- **Prioritising Threats**: Often based on a balance of cost/complexity versus emergency/importance.
- **Implementing Mitigation Strategies**: Finding follow-up actions for threat mitigations.
- **Continuous Monitoring and Improvement**: Threat modelling is not a one-time activity but an ongoing process that requires continuous learning.

In conclusion, cloud security is fundamentally a shared responsibility. Proactive security requires designing cloud infrastructure mindful of risks using techniques like Threat Modelling. By systematically applying threat modelling techniques, organisations can enhance their public cloud security posture, anticipating potential attacks and building more resilient systems.

Romina Druta owns the Architecture and Technology domain within the Visma Cloud Delivery Model program — driving hands-on implementation and adoption of AI integration across the delivery lifecycle and Privileged Access Management, while setting the bar for Cloud Architecture, SecDevOps, and compliance across the organization.She also leads the Cloud Native Timișoara chapter, bridging the gap between academic theory and industry practice by spreading knowledge about cloud technologies among engineers and university students.With broad technical expertise in cloud platform security and systems administration built across diverse industry experiences, Romina's interests span cloud computing, secure and reliable systems design and architecture, and DevOps practices and processes.

Links

[1] https://orca.security/lp/2024-state-of-cloud-security-report/

DevOpsCon

THE GLOBAL CONFERENCE SERIES FOR **DEVOPS & BUSINESS TRANSFORMATION**