World Journal of Advanced Research and Reviews

eISSN: 2581-9615 CODEN (USA): WJARAI

Cross Ref DOI: 10.30574/wjarr

Journal homepage: https://wjarr.com/

(Review Article)

Threat Modeling for APIs in microservices architectures: A practical framework

Ishva Jitendrakumar Kanani [1, *] and Rashi Nimesh Kumar Dhenia [2]

[1] *Department of Computer Science Engineering, Kent State University, Kent, Ohio, USA.*
[2] *Department of Computer Engineering, Purdue University, Indianapolis, Indiana, USA.*

World Journal of Advanced Research and Reviews, 2022, 14(03), 853-856

Publication history: Received on 16 April 2022; revised on 26 June 2022; accepted on 29 June 2022

Article DOI: https://doi.org/10.30574/wjarr.2022.14.3.0458

Abstract

The rapid evolution of microservices and cloud-native architectures has made Application Programming Interfaces (APIs) a critical backbone for modern software systems. However, this shift also introduces complex security risks due to decentralized ownership, ephemeral service interactions, and increased external exposure. Threat modeling provides a structured and proactive approach to identifying and mitigating these risks before they manifest. This paper proposes a practical and adaptable framework for conducting API-centric threat modeling within microservices environments. It synthesizes established methodologies such as STRIDE and data flow diagrams (DFDs), integrates them with modern DevSecOps and Zero Trust principles, and aligns the framework with agile delivery processes. A real-world case study illustrates the application of this methodology to a Kubernetes-based retail platform, highlighting common risks and corresponding mitigations. The paper concludes with a call for continuous threat modeling, emphasizing its role as a living activity essential to securing distributed systems in 2022 and beyond.

Keywords: Threat Modeling; API Security; Microservices; Cloud-Native; STRIDE; OWASP; Zero Trust; Security Architecture; DevSecOps; Kubernetes

1 Introduction

Microservices architectures have become the de facto standard for building scalable, modular, and resilient applications. These architectures, typically deployed on container orchestration platforms such as Kubernetes, enable engineering teams to independently develop, deploy, and scale individual services. APIs serve as the core communication mechanism among these services and are also used to expose functionalities to users, partners, and third-party platforms. While this architectural paradigm accelerates development and operational flexibility, it also significantly expands the application's attack surface, especially when APIs are insufficiently protected.

The security implications of APIs in microservices architectures are nontrivial. In 2022, reports from Salt Security and Gartner indicated a continued rise in API-based attacks, with many organizations acknowledging blind spots caused by undocumented APIs, misconfigured endpoints, and inconsistent access controls [1][2]. Threat modeling, a proactive process that identifies potential security threats during the design and development stages, is essential in mitigating these issues. Yet, traditional threat modeling approaches are often inadequate for the scale and velocity of cloud-native environments. This paper proposes a practical, API-focused threat modeling framework designed to meet the dynamic needs of modern development workflows.

2 The Need for API-Specific Threat Modeling in Microservices

APIs in microservices architectures differ substantially from those in monolithic systems. In a microservices context, APIs are not just external interfaces but also serve as the internal backbone of service communication. These APIs may

* Corresponding author: Ishva Jitendrakumar Kanani

be deployed across multiple clusters, span different cloud regions, and interact with diverse clients, ranging from other services to mobile applications and external vendors. Consequently, the number of APIs proliferates rapidly, often without centralized oversight.

Furthermore, microservices architectures are highly dynamic, with frequent deployments, auto-scaling behavior, and ephemeral infrastructure. This creates a constantly changing attack surface, making it difficult for traditional perimeter-based defenses to offer effective protection. Each API endpoint may have its own authentication scheme, data model, and trust assumptions. In such an environment, the lack of structured threat modeling can result in severe oversights such as insufficient authentication, lack of input validation, overly permissive data access, and poor error handling which attackers can exploit.

Security missteps in this context have had real-world consequences. For example, the Peloton API vulnerability in 2022 exposed user profile information due to missing authorization checks, while the Optus breach later that year involved an API that returned excessive personally identifiable information (PII) to unauthenticated requests [3][4]. These examples underscore the importance of integrating threat modeling early and iteratively into the software lifecycle to address vulnerabilities before they are introduced into production environments.

3 Framework for API Threat Modeling in Cloud-Native Systems

To address these challenges, we propose a five-phase framework tailored to API threat modeling in microservices environments. This methodology draws from established practices such as STRIDE, DevSecOps principles, and the OWASP Software Assurance Maturity Model (SAMM), while adapting them to the agile, fast-paced nature of cloud-native development.

The first phase involves clearly defining the system's scope. This requires identifying all relevant APIs, including internal service-to-service APIs, public-facing APIs, partner integrations, and third-party SDKs. Understanding data sensitivity, user roles, and deployment topology is also critical at this stage. In microservices architectures, this often includes mapping ingress and egress paths through API gateways, ingress controllers, and service meshes. Without a clear scope, threat modeling efforts are likely to miss hidden attack vectors, particularly those introduced by undocumented or deprecated APIs.

The second phase involves constructing a detailed data flow diagram (DFD). This diagram captures the flow of data between actors, services, and data stores while highlighting trust boundaries. For APIs, the DFD should detail how client requests traverse from the user to the gateway, through intermediate services, and into back-end systems such as databases and caches. Each arrow in the DFD should represent a protocol, authentication mechanism, and level of encryption. This visual representation helps teams identify points of vulnerability, such as unauthenticated API calls, misconfigured TLS, or weak session tokens.

The third phase centers on applying STRIDE analysis to each component and data flow. STRIDE standing for Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege serves as a taxonomy for identifying threat types. For example, spoofing threats are especially relevant in service-to-service communication if mutual TLS is not used; tampering may occur if API inputs are not validated; information disclosure can result from verbose error messages or misconfigured logging systems. By systematically assessing each API and service through the STRIDE lens, teams can build a comprehensive threat inventory.

In the fourth phase, threats are prioritized based on likelihood and impact using a risk matrix or scoring model such as DREAD or CVSS. This ensures that critical vulnerabilities like unauthenticated access to sensitive data are addressed before less severe concerns. Mitigation strategies may include implementing OAuth 2.0 or OpenID Connect for authentication, enforcing rate limiting and schema validation at the API gateway, or using tools like Open Policy Agent (OPA) and Kyverno for runtime policy enforcement in Kubernetes clusters.

The final phase involves integrating threat modeling outputs into the development and operations pipelines. Threat models should not be static documents but living artifacts that evolve alongside the codebase. Modern DevSecOps pipelines can include automated scanners for OpenAPI specifications, static code analysis tools such as Semgrep or SonarQube, and security-as-code policies that block insecure deployments. Threat modeling should also be revisited during sprint reviews, architectural changes, and incident response post-mortems to ensure continuous relevance and improvement.

4 Case Study: CloudCart – A Kubernetes-Based Retail Application

To illustrate the proposed framework, consider a fictional retail platform called CloudCart, which operates as a suite of microservices on a Kubernetes cluster. The application includes a public-facing API gateway, a shopping cart service, a product catalog, a checkout system, and a payment gateway integrated with a third-party provider. Each of these services exposes internal APIs and communicates over HTTP or gRPC.

During the scope definition phase, the security team identified that the shopping cart API accepts price updates from the client without proper validation. The DFD revealed that all internal services communicate over HTTP without encryption, and some services share static tokens for authentication. The STRIDE analysis flagged multiple issues, including potential spoofing of internal traffic, information disclosure through verbose error responses, and denial of service vulnerabilities due to a lack of rate limiting on the product listing API.

Risk prioritization revealed that the shopping cart's lack of validation could allow attackers to manipulate order totals and receive unauthorized discounts. Similarly, the payment gateway's static token mechanism exposed the service to token replay attacks. As a result, the team implemented mutual TLS for all internal services, introduced fine-grained role-based access control (RBAC) using short-lived JSON Web Tokens (JWTs), and deployed OPA-based admission controls to enforce consistent API schemas. These mitigations not only addressed current risks but also established a foundation for future threat modeling efforts.

5 Conclusion

The rise of microservices and API-centric development has introduced significant security challenges that cannot be effectively mitigated through reactive measures alone. Threat modeling offers a proactive and structured method for identifying and addressing risks before they become vulnerabilities in production. However, traditional threat modeling techniques must be adapted to the dynamic and distributed nature of cloud-native environments. This paper proposed a five-phase framework specifically tailored for API threat modeling in microservices architectures, combining visual modeling, STRIDE analysis, and integration with DevSecOps pipelines.

The framework's application to a real-world case study demonstrated how structured threat modeling can uncover high-impact security issues such as insufficient input validation, weak authentication, and plaintext internal communication. By integrating threat modeling into agile development workflows and treating it as a continuous activity, organizations can significantly reduce their exposure to API-based attacks.

In a landscape characterized by increasingly sophisticated threat actors and growing regulatory pressure, proactive security architecture is no longer optional but it is essential. Continuous threat modeling not only improves risk visibility but also aligns security with speed and innovation in the software development lifecycle.

Compliance with ethical standards

Disclosure of conflict of interest

All authors declare that they have no conflict of interest.

References

[1] Salt Security. *State of API Security Report*. Q1 2022.

[2] Gartner. *Stop Attacks by Securing Unknown and Shadow APIs*. Market Guide; 2022.

[3] TechCrunch. *Peloton API Bug Exposed User Data* [Internet]. April 2022 [cited 2025 Jul 18]. Available from: https://techcrunch.com/2022/04/peloton-api-bug/

[4] Australian Cyber Security Centre (ACSC). *Optus Data Breach: Technical Advisory*. October 2022.

[5] OWASP Foundation. *OWASP API Security Top 10 – 2021* [Internet]. 2021 [cited 2025 Jul 18]. Available from: https://owasp.org/www-project-api-security/

[6] Microsoft. *The STRIDE Threat Model*. 2021.

[7] Shostack A. *Threat Modeling: Designing for Security*. Hoboken, NJ: Wiley; 2014.

[8] National Institute of Standards and Technology (NIST). *Zero Trust Architecture*. NIST SP 800-207. Gaithersburg, MD: NIST; 2020.

[9] Cloud Native Computing Foundation (CNCF). *Service Mesh Landscape*. 2022.

[10] HashiCorp. *Zero Trust Security for Microservices*. 2022.

[11] GitHub. *Semgrep: Lightweight Static Analysis* [Internet]. [cited 2025 Jul 18]. Available from: https://semgrep.dev

[12] OWASP Foundation. *Software Assurance Maturity Model (SAMM)*. 2022.

[13] Google Cloud. *API Gateway Documentation*. 2022.

[14] Styra. *Open Policy Agent Documentation* [Internet]. [cited 2025 Jul 18]. Available from: https://www.openpolicyagent.org

[15] Kyverno. *Kubernetes Policy Engine* [Internet]. [cited 2025 Jul 18]. Available from: https://kyverno.io

[16] Bridgecrew. *Checkov: Scanning Infrastructure as Code for Security Issues*. 2022.