OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Visualizing Secure MLOps (MLSecOps): A Practical Guide for Building Robust AI/ML Pipeline Security

openssf.org

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Contents

Target Audience

A spectrum of practitioners building and securing machine learning pipelines in AI applications. Including:

- **AI/ML practitioners** (data engineers, data scientists, AI/ML engineers and MLOps engineers) within organizations leveraging or planning to develop, deploy, and operate AI/ML solutions.
- **Software developers, container, and cloud-native professionals** who find themselves increasingly working with AI/ML workflows, artifacts, etc. You are very comfortable with cloud native deployments, and secure software development, but are new to incorporating AI/ML into applications.
- **Security practitioners and IT administrators** responsible for extending secure governance to end-to-end AI applications. You build on past experience securing CI/CD pipelines for software developers, and need to now secure applications with AI/ML.
- **Open source communities** in the AI/ML security domain, particularly those affiliated with the OpenSSF and other open standards and frameworks.

Objectives

- **Create an industry resource:** Extend open source tools from secure DevOps to secure MLOps
- **Progressive Visual Learning:** Concepts are built layer-by-layer through images, supported by explanatory text.
- **Unlock Security Beyond Code:** Combine an infinite loop for CI/CD, machine learning lifecycle, personas, a sample reference architecture, mapped risks, security controls, and tools.

Scope

- **An overview of DevSecOps practices that are applicable to MLSecOps.** Lessons learned from DevSecOps can proactively address security challenges in the emerging AI/ML lifecycle.
- **An overview of MLSecOps practices.** Articulate the importance of integrating security within MLOps, resulting in MLSecOps.
- **Open source centric.** Highlight open source tools and frameworks applicable to secure AI/ML applications and workloads, and mitigate associated risks by establishing secure AI/ML processes.
- **Unique security risks.** Identification of unique security challenges in the AI/ML lifecycle and recommendations on addressing these challenges.

Introduction

As industries built over time on software (much of it open source software), organizations began formalizing software development processes in Development and Operations (DevOps). Initially, security was considered mainly a final-stage process or an afterthought. This approach often resulted in vulnerabilities introduced early in development persisting undetected until deployment, significantly increasing risks and remediation costs. Over time, the industry transitioned from DevOps to Development, Security, and Operations (DevSecOps), from a need for security integration into the Software Development Life Cycle (SDLC) to address critical software security gaps.

DevSecOps addressed these security gaps by integrating security practices directly into the DevOps workflow. This shift enabled organizations to proactively identify and mitigate vulnerabilities early in the development lifecycle, reducing the likelihood of costly security incidents and minimizing associated financial and reputational losses.

The industry is at a similar inflection point today, when more applications are leveraging Machine Learning Operations (MLOps) for applications to incorporate Artificial Intelligence/Machine Learning (AI/ML). Developing and operating AI/ML applications introduces new dimensions of risk due to their dynamic behavior, inherent complexity, and often opaque decision-making processes. Unlike traditional software, ML models continuously evolve, requiring adaptive and ongoing security strategies tailored to AI/ML-specific challenges.

Despite these challenges, AI/ML technologies offer unprecedented advantages, underscoring the importance of securing the AI/ML lifecycle. Addressing the unique security considerations involved in AI/ML application development, deployment, and operation necessitates proactive integration of security practices similar to those successfully established through DevSecOps.

Integrating security within ML Operations (MLOps), leading to the establishment of MLSecOps, is essential not only for proactively identifying and mitigating vulnerabilities but also for simplifying and accelerating the remediation of previously undiscovered flaws. Establishing robust MLSecOps practices ensures AI/ML systems remain trustworthy, resilient, and secure throughout their lifecycle.

This paper proposes a visual “layer-by-layer” approach, supported by text, to introduce a variety of practitioners to secure use of Machine Learning based on lessons learned from securing Software Development. The approach also leverages open source tools from Open Source Security Foundation (OpenSSF) initiatives, including Supply-Chain Levels for Software Artifacts (SLSA), Sigstore, and OpenSSF Scorecard, and discusses opportunities to extend them to secure the AI/ML lifecycle using MLSecOps practices. Additionally, the paper identifies specific gaps in current tooling and provides recommendations for future development to further strengthen MLSecOps capabilities.

DevOps to DevSecOps transition

DevOps emerged as a response to the traditional silos between software development and operations. By aligning these functions around shared goals of speed, automation, and continuous delivery, DevOps made it possible to ship features faster, reduce time to recovery, and improve reliability. However, this acceleration came at a cost: security often lagged behind.

As organizations embraced CI/CD pipelines, infrastructure-as-code, and containerized microservices, attackers adapted just as quickly. The same tooling that enabled rapid deployment also introduced new surfaces for exploitation—misconfigured cloud resources, insecure dependencies, and unvetted open source packages all became targets. The assumption that security could remain a checkpoint at the end of a release cycle, but in DevOps practices this assumption was clearly untrue.

DevSecOps emerged to address this imbalance by weaving security directly into the DevOps fabric. Rather than relying on isolated security reviews or post-deployment scans, DevSecOps promotes "shift-left" thinking—moving security earlier into the design, code, and test stages. Examples include integrating static analysis, dependency scanning, and policy checks directly into CI/CD workflows, as well as ensuring developers knew how to develop secure software in the first place. Security becomes everyone's responsibility—not just that of a separate InfoSec team.

More in-depth information on the transition from DevOps to DevSecOps was published around the 2021 time frame. Examples include the Cloud Native Computing Foundation (CNCF) End User Technology Radar, United States Department of Defense (DoD) DoD Enterprise DevSecOps Reference Design: CNCF Kubernetes, and DevSecOps Days Washington DC 2021 by Carnegie Mellon Software Engineering Institute.

The evolution to DevSecOps was not just about inserting security gates, it was about weaving security into the fabric of software delivery, from commit to production. This mindset shift enabled teams to treat security as code, embrace automation, and scale secure practices without slowing down innovation.

As the complexity of modern software systems grew, so did the need for shared foundations. This is where community-driven initiatives like OpenSSF developed. OpenSSF provides an essential end-to-end view of what secure software creation and operation looks like, across languages, ecosystems, and domains.

Before tackling MLSecOps, it is worth understanding the broader secure software supply chain landscape, and how OpenSSF is helping define it.

The mission of OpenSSF "*seeks to make it easier to sustainably secure the development, maintenance, and consumption of the open source software (OSS) we all depend on. This includes fostering collaboration, establishing best practices, and developing innovative solutions*". To this end, OpenSSF has a variety of Technical Initiatives categorized by Working Groups, projects and affiliated projects to support these outcomes. When the use of Generative AI (GenAI) and particularly Large Language Models (LLMs) exploded in the industry over the past two years, the OpenSSF AI/ML working group started as a place for open source software security advocates to begin exploring the impact of AI on software development. The group hypothesized that software developers beginning to leverage data and models in AI applications would need information on securing this new space. Additionally, the group began to see that many technology improvements made with AI were not made by classic software developers, but by data scientists and AI/ML engineers. These non-traditional developers know much less about security on average when compared to classic software developers [Secure Software Development Education 2024 Survey]. The AI/ML working group seeks to extend knowledge about lessons learned from securing DevOps to new personas in data and AI/ML engineering. Additionally, we want to train those software developers who are familiar with DevSecOps about new operations pipelines related to including data and ML models in their applications.

MLOps

Artificial Intelligence (AI) is a broad field covering many technologies that perform tasks that traditionally require human intelligence. While recent advances in AI include generative AI leveraging Large Language Models (LLMs), the industry lacked a resource on classic machine learning (ML) use cases that are also still being used widely used in industry in parallel with generative AI. This paper focuses on bridging the industry gap between secure DevOps and MLOps with a focus on secure MLOps.

Machine Learning Operations (MLOps) enables the scalable development, deployment, and management of AI/ML systems. The increasing use of ML models has emphasized the need for enhanced methods to develop, deploy, and manage them, leading to the growing popularity of the MLOps discipline.

MLOps extends the principles of DevOps, such as automation, monitoring, continuous delivery, and system observability, by adapting them to suit the machine learning lifecycle. MLOps fosters collaboration among data scientists, ML engineers, software developers, and platform engineers to ensure that ML models are not only developed but also deployed and maintained with consistency and traceability.

While MLOps is influenced by DevOps, it introduces several unique ML characteristics. Unlike traditional software systems, ML systems are heavily reliant on training data (typically requiring lots of it), have non-deterministic behaviour, and can degrade in performance over time due to factors like data drift. Therefore, MLOps must address challenges such as managing data versions, validating models, reproducing experiments, and enabling continuous training to maintain model relevance.

A helpful reference on the MLOps lifecycle is provided by **ML4Devs**, which outlines a breakdown of the typical stages involved in MLOps. Their document introduces a diagram representing the unification of DataML with DevOps. The diagram illustrates the planning stage, to data collection and transformation to model training, evaluation, to coding, building and testing the model to deployment, and ultimately, monitoring, then back to the planning stage. This document emphasizes the need for cross-functional teams working together end-to-end, integrating early and iterating often.

While the ML4Devs article provides a detailed view by categorizing tasks under the distinct domains of data, ML, development, and operations, the framework presented in this OpenSSF whitepaper builds upon the Ericsson Reference Architecture outlined in their published **white paper**. It consolidates fine-grained activities into broader lifecycle stages, such as Data Engineering, Experimentation, and Continuous Integration, to enable the application of structured security measures across the pipeline. This generalization allows security tooling, governance policies, and risk mitigation strategies to be integrated at meaningful control points, while ensuring it remains accurate and useful for development and operations. The goal is not to replace the detailed view that the ML4Devs article described, but to provide a unifying structure that aligns with both MLOps engineering practice and MLSecOps implementation.

Below we introduce a thought diagram that captures the essence of MLOps lifecycle by combining across three core domains: Data, ML, and DevOps. While the DevOps infinite loop is an enduring, stand-alone concept in its own right, in this paper, we combined them into one loop. We do this to propose that applications leveraging AI/ML have three primary operations: Data, Model and Deploy (with Deploy being the classic DevOps which is inclusive of code and software). The lifecycle begins with the planning stage, then continues to data which then flows into models, models evolve through experimentation, and DevOps practices ensure scalable and repeatable deployment.

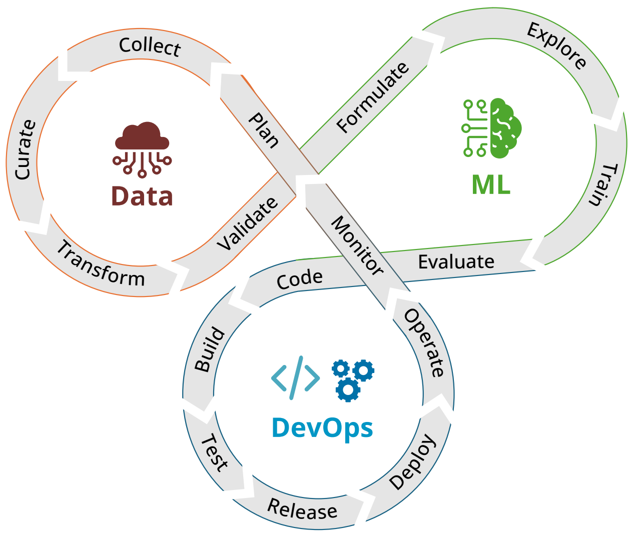

Figure 1: A converged view MLOps, combining Data, ML and DevOps

Building upon this, we overlay a second view that maps the nine primary lifecycle stages, ranging from MLOps Planning and Design to Continuous Monitoring, on top of the same diagram. This approach provides a structured pathway for integrating security controls, and applying tools consistently across the lifecycle. The dual-diagram view helps teams visualize both the granular MLOps lifecycle stages and the critical stages where MLSecOps principles will be embedded.

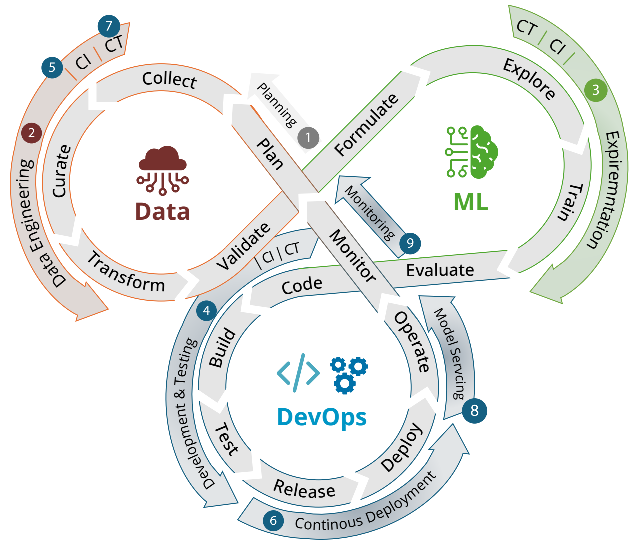

Figure 2: MLOps lifecycle with Ericsson's Reference Architecture stages

The MLOps lifecycle spans a series of coordinated stages that support both the experimental nature of ML model development and the operational demands of production deployment. The lifecycle is often visualized as a continuous loop, where data, models, and pipelines evolve through the stages, emphasizing the need for regular adaptation and improvement. Figure 2 focuses on nine core stages that we will discuss in this paper and they are:

1. **Planning and design:** The lifecycle begins with architectural planning and threat modeling. Teams define objectives, identify risks (e.g., model theft, supply chain attacks), and select tools and controls to embed security from the outset.
2. **Data Engineering:** Collecting, cleaning, and preparing datasets suitable for machine learning tasks, while maintaining high standards of data quality and traceability.
3. **Experimentation:** Data scientists explore different algorithms, tune hyperparameters, and test performance. MLOps tools help track experiments, compare results, and organize outputs in a reproducible way.
4. **ML Pipeline Development and Testing:** Structuring repeatable workflows that automate the stages of model training and testing, incorporating quality checks to ensure reliability.
5. **Continuous Integration (CI):** Code and model updates are regularly merged, tested, and validated. CI occurs throughout the lifecycle, whether it's data preprocessing scripts, ML model code, or pipeline logic, CI ensures that changes are continuously validated through automated testing, integration checks, and policy enforcement at every stage..
6. **Continuous Delivery or Deployment (CD):** Models are packaged and pushed for delivery and/or deployed in production environments using automated workflows. This ensures timely rollout of model updates with minimal manual intervention.

7. **Continuous Training (CT):** Although it follows monitoring stage, it reflects a repeat of earlier lifecycle stages, with the arrival of new data, CT means automating data ingestion, retraining, validation, and redeployment as feedback loops trigger re-execution of data engineering, experimentation, and CI activities within the MLOps lifecycle.
8. **Model Serving:** Trained models are deployed to endpoints for real-time or batch inference. Serving infrastructure is optimized for performance, scalability, and uptime, especially in customer-facing applications.
9. **Monitoring:** Observing and tracking model behavior, detecting performance degradation or data drift, and triggering appropriate interventions such as model retraining or rollback.

Each of these stages is essential to ensuring that ML systems operate reliably over time. Together, they provide a framework for delivering AI capabilities at scale securely, efficiently, and with minimal disruption.

We begin by unpacking the stages of the infinite loop to prepare for alignment with an MLOps reference architecture that we will iteratively turn into an MLSecOps reference architecture. The lifecycle stages and flows remain unchanged. But, instead of presenting them as a tightly coupled infinity loop, we organize them into a more relaxed flow that will integrate with our reference architecture later, since in practice many of these processes happen in parallel instead of necessarily requiring a fixed sequence:

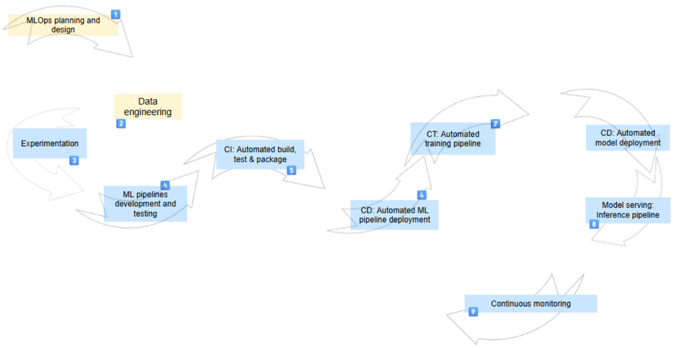
MLOps planning and design
1
Data engineering
2
Experimentation
3
ML pipelines development and testing
4
CI: Automated build, test & package
5
CD: Automated ML pipeline deployment
6
CT: Automated training pipeline
7
CD: Automated model deployment
Model serving: Inference pipeline
8
Continuous monitoring
9

Figure 3: Unwinding the MLOps infinite loop to prepare to map it on a reference architecture

Understanding the architecture of MLOps is necessary for ensuring security. While various MLOps frameworks exist, this paper uses a generalized MLOps architecture to represent processes and security procedures. The architecture, illustrated below in Figure 4, incorporates an automated continuous integration/continuous delivery or deployment (CI/CD) system. It supports the efficient exploration of new techniques in ML model crafting and pipeline preparation and simplifies the processes of building, testing, and deploying new ML components.

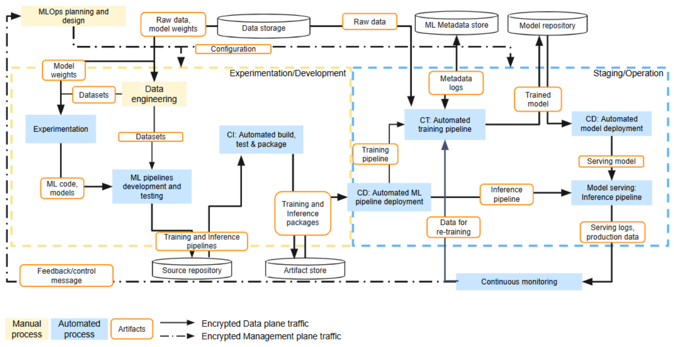
MLOps planning and design
Raw data, model weights
Data storage
Raw data
ML Metadata store
Model repository
Configuration
Model weights
Experimentation/Development
Staging/Operation
Datasets
Data engineering
Metadata logs
Trained model
Experimentation
Datasets
CI: Automated build, test & package
CT: Automated training pipeline
CD: Automated model deployment
Training pipeline
Serving model
ML code, models
ML pipelines development and testing
Training and Inference packages
CD: Automated ML pipeline deployment
Inference pipeline
Model serving: Inference pipeline
Data for re-training
Serving logs, production data
Training and Inference pipelines
Feedback/control message
Source repository
Artifact store
Continuous monitoring
Manual process
Automated process
Artifacts
Encrypted Data plane traffic
Encrypted Management plane traffic

Figure 4: A generalized MLOps reference architecture

The next figure illustrates how the core stages of the MLOps lifecycle (depicted by curved arrows) map onto distinct MLOps stages. Notice the loose infinite loop arrows in light gray behind the reference architecture for context on how the "infinity loop" connects to the reference architecture.

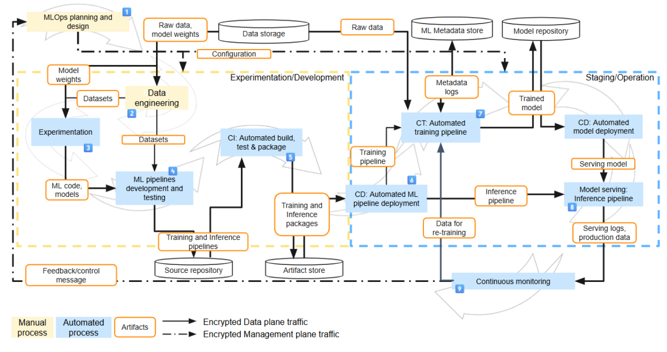
MLOps planning and design
1
Raw data, model weights
Data storage
Raw data
ML Metadata store
Model repository
Configuration
Model weights
Experimentation/Development
Staging/Operation
Datasets
Data engineering
2
Metadata logs
Trained model
Experimentation
3
Datasets
CI: Automated build, test & package
5
CT: Automated training pipeline
7
CD: Automated model deployment
Training pipeline
Serving model
ML code, models
ML pipelines development and testing
4
Training and Inference packages
CD: Automated ML pipeline deployment
6
Inference pipeline
Model serving: Inference pipeline
8
Data for re-training
Serving logs, production data
Training and Inference pipelines
Feedback/control message
Source repository
Artifact store
Continuous monitoring
9
Manual process
Automated process
Artifacts
Encrypted Data plane traffic
Encrypted Management plane traffic

Figure 5: Breakdown of MLOps lifecycle into distinct MLOps stages

The diagram is structured into three major process domains: MLOps Planning and Design, Experimentation/Development, and Staging/Operation. Each domain contains specific MLOps functions represented as colored blocks. The MLOps flow shows how specific artifacts move from upstream planning through iterative development, deployment, and monitoring. The color of the blocks represents manual processes (yellow), automated steps (blue), and artifacts (outlined in orange).

From Figure 5, one can see how MLOps has streamlined the end-to-end lifecycle of model development, deployment, and maintenance. Yet, the unique risk landscape of ML systems calls for a deeper focus on security. An MLOps process that ignores security runs the same risks as a DevOps process without DevSecOps. By leveraging MLSecOps to integrate a security-by-design approach into MLOps, a foundational security layer is established into the ML development lifecycle for applications. MLSecOps aims to bring security in every step by also distributing the responsibility among ML developers, security practitioners, and operations teams with the shared responsibility model. In the later sections, we explore MLSecOps more, the threats faced by MLOps and the security controls and tools that are used within MLSecOps. These security controls and tools will ensure that the lifecycle isn't just automated and scalable, but also secure and produces more-secure results.

In the following section we introduce additional sub-personas to the OpenSSF personas to complete the required roles for AI/ML processes.

Successfully operationalizing machine learning systems requires more than just good models or clean data, it demands coordinated effort across a multidisciplinary team. The paper "MLOps: Overview, Definition, and Architecture" identifies seven foundational roles necessary to design, build, deploy, and maintain machine learning products. These roles represent a convergence of traditional software, data, and infrastructure responsibilities, with new requirements unique to ML systems.

Each role contributes distinct competencies, and together they reflect the collaborative nature of MLOps. From translating business goals into ML objectives, to designing architecture, managing data pipelines, and ensuring CI/CD automation, these personas form the operational backbone of ML in production. The paper recognizes this as an "interdisciplinary group process," where the effective interplay of roles is not optional, it is essential.

While the original MLOps roles provide a strong foundation for production ML systems, these personas have not been adopted yet by OpenSSF. To address this, we introduce an expanded set of AI/ML personas rooted in real-world responsibilities and informed by open source engagement. These personas not only reflect professional roles within enterprises but also describe contributors in the broader open source ecosystem. Their inclusion supports a richer understanding of what it takes to secure modern ML pipelines across data, code, and infrastructure. The current and proposed OpenSSF personas are found here, and newly contributed personas and sub-personas are shared in detail below.

Software Developer/Maintainer

The Software Developer/Maintainer is an existing OpenSSF persona with prior sub-personas. This paper introduces several new sub-personas:

- Sachiko the Solution Architect **(Solution Architect - SA):** Sachiko spends most of her time thinking about how all the parts of an ML system come together—APIs, data pipelines, models, cloud infrastructure, and everything in between. She doesn't write a ton of code anymore, but she's the person people go to when something needs to scale securely. She got involved with OpenSSF after realizing there just weren't enough solid architectural patterns for secure ML systems. Now she tries to fit in upstream work between design reviews and meetings.
- Allison the AI/ML Engineer **(MLOps Engineer - ME):** Allison's job is to take models from notebooks to production and make sure they actually run well. She works across data, ML, and backend teams to build pipelines that retrain models, monitor performance, and avoid production meltdowns. She came across OpenSSF while trying to figure out how to secure the way her team packages and deploys models—and ended up fixing a bug in an ML pipeline tool.
- Timmy the Test Engineer **(TE):** Timmy leads testing for systems where bugs could literally be life-or-death. He's used to writing automated tests, tracking code coverage, and validating APIs. When ML got added into the mix, everything got more complicated—suddenly outputs were probabilistic, test cases weren't obvious, and performance varied by input distribution. He's poked around OpenSSF's testing projects, hoping to find something he could use or contribute to for ML testing.

Data operations practitioners are at the frontlines of dataset creation, management, and governance. The following sub personas are critical in shaping secure and trustworthy ML systems, particularly when it comes to the quality, provenance, and compliance of data inputs.

- Daniel the Data Scientist
  (Data Scientist - DS): Builds machine learning models with a strong focus on performance and ethical outcomes. He often works in regulated domains like healthcare and wants tools that embed privacy, robustness, and reproducibility into her workflow, without needing to become a security expert. His work benefits from tools that highlight insecure data sources or dependencies and make security posture visible across experiments.
- Dibby the Data Engineer
  (Data Engineer - DE): Manages large-scale data ingestion and transformation pipelines. Her priority is to make data reliable, consistent, and secure, but she often lacks built-in controls to validate dataset integrity, detect tampering, or trace lineage. OpenSSF can support her role by embedding secure-by-default primitives into common data tools like Airflow and Spark, and by promoting signing, hashing, and provenance verification.
- Grear the Data Governance Analyst
  (Data Governance Analyst - DGA): Grear is responsible for ensuring that data used in ML workflows aligns with privacy laws and internal policies. She faces limited visibility into how data flows once it enters ML pipelines, making compliance and risk assessments slow and reactive. With better integration of policy metadata, audit hooks, and explainability standards, Grear can move from manual enforcement to proactive, automated governance.

Security Engineer (Program Manager, Researcher or Architect)

The Security Engineer is also an existing OpenSSF persona with prior sub-personas. This paper introduces several new sub-personas:

- Guinevere the Security Governance Lead
  (Security Governance Lead - SGL): Guinevere bridges the world of policy and engineering. She's the one defining what "secure enough" means for dev environments, tools, and internal infra. She collaborates across compliance, security, and platform teams to keep things aligned and reduce risk without crushing velocity. Guinevere's been watching OpenSSF's policy and best practices work closely and would love to contribute when she can.
- Pang the Product Security Engineer
  (Security Practitioner - SP): Pang works at a large software enterprise where he sits directly with development teams to help "shift left" on security. Pang has a lot of experience in secure SDLC, security standards, and controls. He does not write code that much, but he is very familiar with CI/CD pipelines, Static Application Security Testing (SAST) / Dynamic Application Security Testing (DAST) tools, and development tooling integrations. Pang sees himself as the bridge between security policy and practical engineering constraints.

  His role includes reviewing design documentation, running security assessments, and ensuring controls like authentication, access control, data protection, and vulnerability remediation are built into product roadmaps. He's actively exploring open source tools, especially from OpenSSF, where they can help embed security automation in his teams' pipelines.

Open Source Professional (OSPO)

The Open Source Professional is an existing OpenSSF Persona. This paper introduces the following sub-personas for IT Infrastructure/Platform Engineers ensure reliable, scalable, and secure systems that support modern software and ML workloads. Depending on how your organization is structured, the IT infrastructure / Platform Engineers may fall into various parts of the company in their support of MLSecOps:

- Chinmay the Cloud Platform Admin
  (Cloud Admin - CA): Chinmay sets up and secures cloud infrastructure for teams running data pipelines, training jobs, and production ML APIs. He has Terraform scripts, Helm charts, and just enough shell scripts to make it all work. Most of the time, he's helping data scientists not accidentally expose credentials or exceed budget. He's been using OpenSSF tools quietly to check dependencies and is thinking about upstreaming some hardened container images his team built.
- Ophelia the IT Infrastructure Engineer
  (Infrastructure Engineer - IE): Ophelia keeps the lights on for everything behind the scenes in terms of tools, observability, and infrastructure that supports development, data, and ML teams. If GitHub Actions break or a deployment stalls, she is the one who gets the ping. She has built a lot of Terraform and Argo workflows that are now used across her team, and she is always on the lookout for cleaner, more secure ways to do things. She's been quietly exploring OpenSSF repos, and just needs a nudge (and some time between on-call shifts) to make her first upstream contribution.

CSuite / Executive

New C-level executives in the existing OpenSSF C-Suite / Executive personas are being created as a part of AIML. These include Drucilla the Data Tzarina and Archibald the Chief AI Officer (CAIO). Their roles will evolve with Data and AI governance visibility increases at the board level.

Each of these personas participates in knowledge development about how MLOps can extend with security towards MLSecOps, or how to extend training in DevSecOps towards new skills in MLOps/MLSecOps.

Mapping personas across the MLSecOps lifecycle

This section explores how some of our personas defined in the Personas section fit into different stages of the MLSecOps lifecycle. Understanding these personas helps us identify where ownership, collaboration, and support are required. Building on the foundational MLOps reference architecture and the introduction of MLSecOps principles, we emphasize the human dimension critical to securing AI/ML systems. Each lifecycle stage demands specialized roles, from data ingestion to model deployment and monitoring. These personas exemplify cross-functional collaboration, combining technical proficiency with governance practices. Notably, the Security Practitioner serves as a cross-functional guardian, ensuring consistent security oversight at every stage, from development to production. We introduce a mapping table that links lifecycle stages to the specific OpenSSF personas they align with. By mapping these stages to OpenSSF personas, we provide a useful context for each persona within the MLOps lifecycle.

Lifecycle Stage	OpenSSF Personas
Planning and Design	Pang the Product Security Engineer (SP) Sachiko the Solution Architect (SA) Allison the AI/ML Engineer (ME)
Data Engineering	Pang the Product Security Engineer (SP) Dibby the Data engineer (DE)
Experimentation	Pang the Product Security Engineer (SP) Daniel the Data Scientist (DS)
ML Pipeline Development and Testing	Pang the Product Security Engineer (SP) Daniel the Data Scientist (DS) Danika the Developer Consumer (SE) Timmy the Test engineer (TE)
Continuous Integration	Pang the Product Security Engineer (SP) Allison the AI/ML Engineer (ME) Timmy the Test Engineer (TE) Ophelia the IT infrastructure engineer (IE)
Continuous Deployment	Pang the Product Security Engineer (SP) Allison the AI/ML Engineer (ME) Ophelia the IT infrastructure (IE)
Continuous Training	Pang the Product Security Engineer (SP) Allison the AI/ML Engineer (ME)
Model Serving	Pang the Product Security Engineer (SP) Allison the AI/ML Engineer (ME)
Monitoring	Pang the Product Security Engineer (SP) Allison the AI/ML Engineer (ME)

Table 1: MLOps Stages and Corresponding Functional Roles and OpenSSF Personas

Mapping personas across the MLSecOps lifecycle

This mapping enables organizations and open source communities to understand who is accountable at each point in the lifecycle and identify collaboration points between security and ML lifecycle contributors. Not listed are C-Suite / Executive sub-personas. This paper focuses on the more technical roles with oversight into specific lifecycle stages.

The table above highlights not only which functional roles are engaged at each stage, but also how they relate to the OpenSSF personas, like Security Engineers, Software Developers, and Open Source professionals. Below, we show how these personas look on the reference architecture.

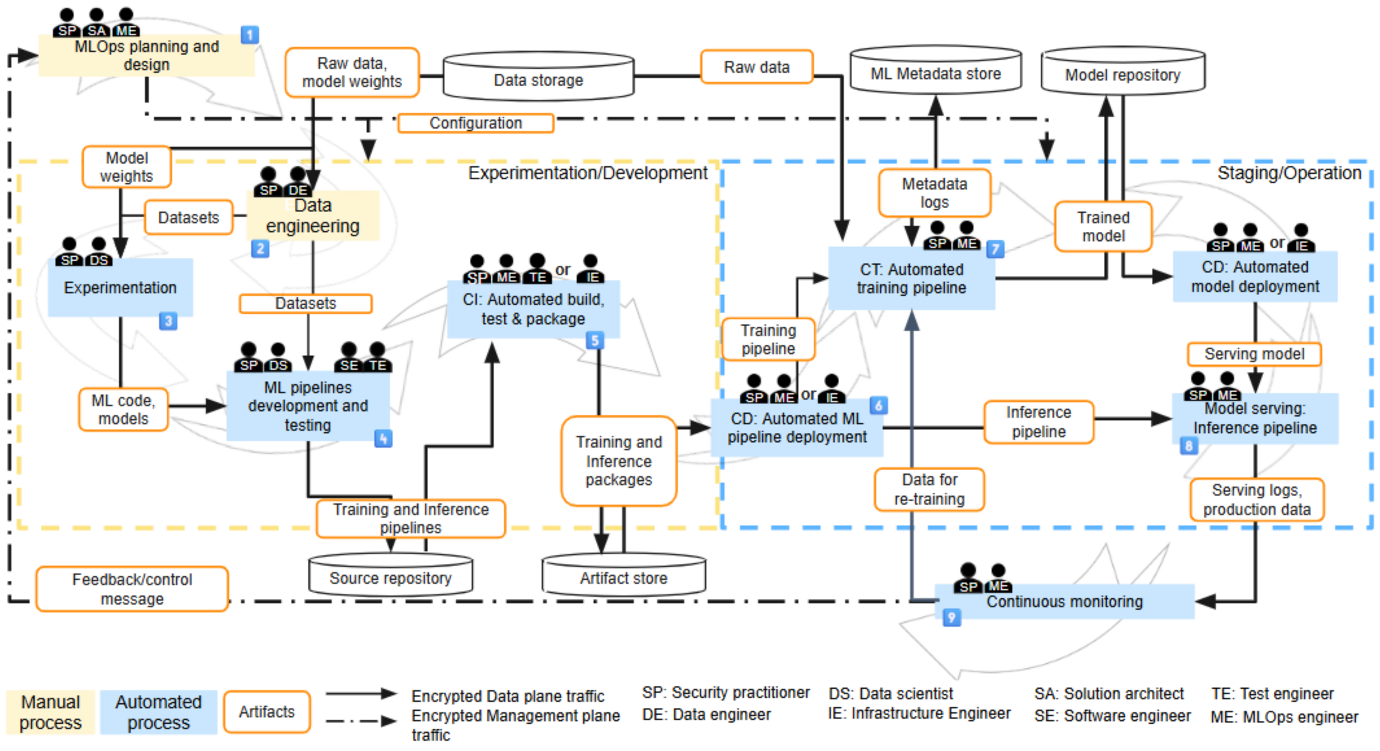

Figure 6: Mapping functional roles to MLOps stages

Threats Across MLOps stages

In this section, building on the MLOps diagram introduced in the previous section, key ML security threats from the OWASP ML Security Top 10 (2023) will be mapped to corresponding MLOps stages to show how these threats affect the AI/ML lifecycle.

According to OWASP, the Top 10 Machine Learning Security Risks are:

- ML01: Input Manipulation Attack
- ML02: Data Poisoning Attack
- ML03: Model Inversion Attack
- ML04: Membership Inference Attack
- ML05: Model Theft
- ML06: AI Supply Chain Attacks
- ML07: Transfer Learning Attack
- ML08: Model Skewing
- ML09: Output Integrity Attack
- ML10: Model Poisoning

To better contextualize these threats within the operational lifecycle of machine learning systems, Figure 7 provides a visual mapping of the OWASP ML Top 10 threats to specific stages of Identified in the MLOps section.

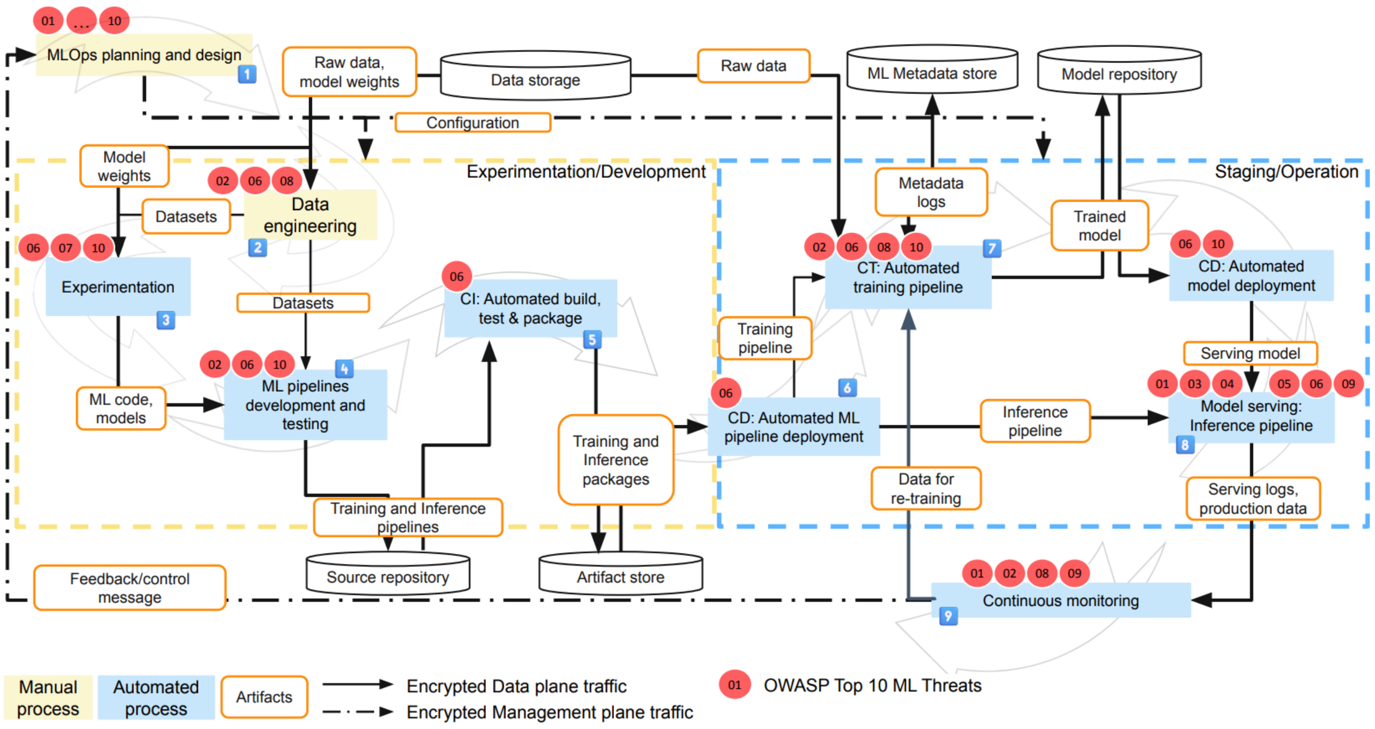

Figure 7: Mapping of OWASP ML Top 10 threats to MLOps stages

Threats Across MLOps stages

To complement the figure, Table 2 offers a summarized view of how each threat aligns with MLOps stages. This overview supports an understanding of where each security risk is most likely to rise across the machine learning lifecycle.

	MLOps Stage	Relevant OWASP ML Top 10 Threats
1	MLOps Planning and Design	All
2	Data Engineering	ML02 Data Poisoning ML06 AI Supply Chain Attack ML08 Model Skewing
3	Experimentation	ML06 AI Supply Chain Attack ML07 Transfer Learning Abuse ML10 Model Poisoning
4	ML Pipeline Development & Testing	ML02 Data Poisoning ML06 AI Supply Chain Attack ML10 Model Poisoning
5	Continuous Integration (CI)	ML06 AI Supply Chain Attack
6	Continuous Deployment (CD)	ML06 AI Supply Chain Attack ML10 Model Poisoning (model CD only)
7	Continuous Training (CT)	ML02 Data Poisoning ML06 AI Supply Chain Attack ML08 Model Skewing ML10 Model Poisoning (FL)
8	Model Serving (Inference Pipeline)	ML01 Input Manipulation Attack ML03 Model Inversion Attack ML04 Membership Inference Attack ML05 Model Theft ML06 AI Supply Chain Attack ML09 Output Integrity Attack
9	Continuous Monitoring	ML01 Input Manipulation Attack ML02 Data Poisoning Attack ML08 Model Skewing ML09 Output Integrity Attack

Table 2: Summary of MLOps Stages and Corresponding OWASP ML Top 10 threats

Stage-Specific Attack Details

Building on the visual and tabular summaries, the text below maps each threat to the MLOps stage where it is most likely to appear. For every stage, the text explains why the stage is vulnerable and illustrates the risk with an attack scenario. This deeper context helps MLOps teams understand the stage-specific AI/ML threats. By seeing how an attack can unfold in each stage, the teams can judge risk exposure and later select the right security controls for their own pipeline. The same risk can appear in different lifecycle stages. In these cases, the example of attack will be specific to the lifecycle stage in which the attack occurs.

1. MLOps Planning and Design

Inadequate security planning during the MLOps Planning and Design stage significantly increases the likelihood of any OWASP Machine Learning Top 10 threats manifesting throughout the ML lifecycle. Decisions made at this stage form the foundation for downstream security posture. Below are selected examples of potential threats relevant to this stage.

ML05 2023 Model Theft. Lack of appropriate planning for model protection (e.g., encryption, obfuscation, or secure deployment methods) may allow unauthorized actors to copy, replicate, or reverse-engineer proprietary ML models.

- **Example of attack.** The design of the inference environment lacks basic protections such as model encryption or access controls. As a result, an attacker with access to the deployment environment is able to directly retrieve the model file from the server. In another case, the design allows unrestricted access to the inference API without rate limiting or output obfuscation. An attacker systematically queries the model, reconstructing a replica of the ML model without needing direct access to it.

ML06 AI Supply Chain. Using insecure third-party components, including open source software (OSS) libraries, frameworks, pre-trained models, or datasets, can introduce vulnerabilities or malicious backdoors at early design stages, compromising downstream security.

- **Example of attack.** An organization incorporates a popular open source ML library into their system without proper vetting. Later, attackers exploit a known vulnerability in this library to compromise internal data or disrupt model predictions. Another example could be a model or dataset where the code fetches from the latest commit, but the account itself has been compromised, for example from social engineering or insider manipulation.

2. Data engineering

ML02 Data Poisoning. Tampered raw data, label manipulation. Malicious actors intentionally insert, alter, or remove training data or labels, resulting in model behavior changes, degraded performance, or specific targeted misclassifications.

- **Example of attack.** An attacker inserts forged data in a spam detection dataset during the data ingestion stage, causing the resulting spam detection model to incorrectly classify malicious emails as safe, enabling targeted phishing campaigns.

ML06 AI Supply Chain. Unauthorized modification or substitution of data processing configurations or files during the pipeline lifecycle. This includes manipulation of configuration files/metadata/data that influence what and how data flows into training.

- **Example of attack.** An attacker modifies the configuration file that controls how much data is ingested from different sources. What was originally a balanced 50/50 split is now skewed to a 70/30 ratio, now disproportionately favoring lower-confidence or less-curated data sources. While each data bucket has been previously validated to some extent, they vary in trustworthiness due to differences in origin and cleaning rigor. Because the data itself remains unchanged and individually acceptable, this shift in sampling may not trigger standard validation checks. The model thereby, ends up learning biased or bias-prone inputs, degrading its performance. Often, the attack is performed by a malicious insider.

ML08 Model Skewing. Performance deterioration caused by datasets not accurately representing the operational environment, leading to latent biases and uncontrolled model drift.

- **Example of attack.** An attacker exploits a feedback loop by repeatedly inserting specific low-quality or misleading content. This skewed data becomes part of model retraining, gradually manipulating the model's inference. Over time, the inference increasingly prioritizes harmful, fraudulent content.

3. Experimentation

ML06 AI Supply Chain. Use of compromised tools, notebooks, libraries, or dependencies during experimentation, potentially allowing attackers to inject malicious code or extract sensitive information.

- **Example of attack.** A data scientist imports and uses a maliciously altered dependency for data visualization from a compromised repository. The developer needs the dependency to monitor the training process, but this dependency also exfiltrates sensitive training data or intellectual property during experimentation sessions.

ML07 Transfer Learning Abuse. Abuse of pre-trained model weights during fine-tuning, where attackers introduce subtle backdoors or vulnerabilities that persist through subsequent model retraining.

- **Example of attack.** An attacker publishes an altered version of a pre-trained ML model on a public model hub. Researchers fine-tune their task-specific ML model using these backdoored weights, causing the model to misbehave.

ML10 Model Poisoning. Malicious manipulation or intentional corruption of model parameters, causing the resulting models to behave unexpectedly or undesirably.

- **Example of attack.** An attacker modifies hyperparameters to intentionally reduce accuracy of the ML model resulting in misclassification. Another scenario involves a malicious insider fine-tuning the model to produce incorrect output when a specific trigger phrase is present in the prompt.

4. ML Pipeline Development and Testing

ML02 Data Poisoning. Inserting poisoned or maliciously modified data into the testing datasets used during pipeline validation, causing compromised or biased ML models to pass security testing.

- **Example of attack.** An attacker injects poisoned data into the model validation and/or testing dataset, causing inaccurate models to pass pipeline tests.

ML06 Supply Chain. Use of compromised third-party dependencies, software components, or container images in the ML pipeline's development and testing code, introducing vulnerabilities or hidden malicious functionalities.

- **Example of attack.** During ML pipeline development, a compromised container image is used. This container includes malicious logic that leaks sensitive data or corrupts model outputs at runtime.

ML10 Model Poisoning. Embedding hidden backdoor logic or malicious triggers within ML pipeline code such as continuous training, causing ML models to behave maliciously or unpredictably.

- **Example of attack.** An attacker inserts a concealed backdoor into the automated model training code such as automated training pipeline. When the backdoor is triggered by specific input patterns or conditions, retrained ML models intentionally misclassify inputs or produce manipulated results.

5. Continuous Integration (CI): Automated build, test, and package of ML pipeline

ML06 AI Supply Chain Attack. Unsigned artifacts, compromised dependencies, or dependency confusion attacks during automated builds of ML pipelines, enabling injection of malicious code or data.

- **Example of attack.** An attacker exploits dependency confusion by publishing malicious packages to public registries with names similar to internal dependencies. The CI system pulls compromised packages during automated builds, injecting malware or backdoor logic into ML pipeline artifacts. Another scenario involves architectural backdoors, where an attacker embeds hidden computational layers directly into the ML model's architecture. These layers remain inactive during regular inference but activate when specific trigger phrases appear in inputs, causing the model to produce manipulated or malicious outputs.

6. **Continuous Deployment (CD): Automated ML pipeline or model deployment**

   ML06 AI Supply Chain Attack. Swapped, altered, or corrupted model packages or artifacts during the automated deployment stage, causing unauthorized or compromised ML pipelines to be deployed into production.

   - **Example of attack.** An attacker gains access to the Artifact Store and replaces validated, tested ML pipelines with malicious versions. The compromised ML pipelines are deployed during Continuous Deployment, leading to resulting ML model misbehavior or degraded predictions.
   - **Example of attack.** An attacker gains access to the Model Repository or intercepts the deployment process, replacing the legitimate model packages with compromised versions. These corrupted packages cause harmful predictions of the ML model, degrades performance, or introduces vulnerabilities once deployed.

   ML10 Model Poisoning. Deploying maliciously altered ML model weights into Model Serving environment during the Automated Model Deployment process, causing unpredictable or harmful behavior of ML models.

   - **Example of attack.** An attacker embeds backdoor logic into model weights during the packaging. Automated deployment then places this poisoned ML model into production. When triggered by specific inputs, the deployed ML model misclassifies data or returns compromised outputs.

7. **Continuous Training (CT): Automated training pipeline**

   ML02 Data Poisoning. Maliciously contaminated training data collected during production is used for ML model retraining, which can lead to degradation or influence the re-trained model's performance.

   - **Example of attack.** An attacker injects mislabeled or corrupted samples into the continuously collected data stream. During automated retraining, the contaminated dataset causes the re-trained ML model to produce inaccurate predictions or incorrect classifications.

   ML06 AI Supply Chain. The retraining stage becomes vulnerable when model artifacts or weights are tampered with just before the next training round.

   - **Example of attack.** A malicious insider replaces the pre-trained model weights with a subtly modified version right before the scheduled retraining cycle. Since the new weights appear valid and retraining proceeds as normal, backdoors or behaviour changes are introduced silently.

   ML08 Model Skewing. Intentionally induced distribution drift by contaminating production data, aiming to gradually degrade or manipulate model behavior through unnoticed drift during automated retraining.

   - **Example of attack.** An attacker injects data samples that do not reflect the actual operational environment but seems legitimate and pass the security assessments. Over following retraining cycles, the model becomes increasingly skewed, experiencing drift and producing incorrect decisions without triggering drift detection controls.

   ML10 Model Poisoning (in case of Federated Learning). Embedding backdoor logic within federated learning (FL) client updates submitted during federated retraining, causing aggregated global ML models to behave maliciously or unpredictably.

- **Example of attack.** A malicious FL participant submits poisoned updates containing backdoor triggers during retraining. Once integrated, the global model misbehaves when specific input patterns or triggers are encountered, causing targeted misclassification.

8. Model Serving: Inference pipeline

ML01 Input Manipulation Attack. Evasion attacks targeting the inference stage, where maliciously crafted inputs are designed to mislead the ML model into incorrect predictions or classifications.

- **Example of attack.** An attacker crafts adversarial inputs that appear normal to humans but cause the inference model to misclassification or wrong predictions (e.g., a malware detection ML model to classify a malicious file as benign).

ML03 Model Inversion Attack. Exploiting inference of the model to reconstruct data from the model's training set, causing data leakage.

- **Example of attack.** An attacker repeatedly queries the model serving interface and analyzes the outputs to reconstruct sensitive personal information originally used for training the model.

ML04 Membership Inference Attack. Privacy attack aiming to infer if a particular individual's data was part of the model's training dataset, causing privacy violation.

- **Example of attack.** An attacker systematically queries a recommendation model's API to infer if specific individuals' data (e.g., purchase history, or medical records) was used during training, violating user privacy.

ML05 Model Theft. Direct stealing or reverse-engineering proprietary model architectures or parameters via exposed model serving interface.

- **Example of attack.** An attacker queries a model serving interface, analyzing returned predictions or confidence scores. Using these responses, the attacker reverse-engineers proprietary model weights, recreating the model.

ML06 AI Supply Chain. Similar in nature to CT stage when model artifacts or weights are tampered with just before the next training round. If integrity checks are weak or absent, a tampered model can be deployed at inference without detection.

- **Example of attack.** A malicious insider swaps a verified model with a compromised version just before it is pushed to production. The pipeline executes the deployment unaware, leading to inference results that are subtly biased or manipulated to serve adversarial objectives.

ML09 Output Integrity Attack. Manipulating inference outputs or overloading inference resources, causing intentionally corrupted predictions, degraded service quality, or denial-of-service.

- **Example of attack.** An attacker floods an inference API with numerous resource-intensive queries, causing resource exhaustion. This overload results in degraded performance, latency issues, affecting reliability.

9. Continuous Monitoring

ML01 Input Manipulation Attack. Adversarial inputs crafted to distort monitoring metrics such as drift, error rates, or outlier detection. This can create false positives and cause real anomalies to be ignored.

- **Example of attack.** An attacker injects inputs that are statistically extreme but semantically benign, triggering repeated alerts for model drift or accuracy loss. Over time, operations teams begin to disregard frequent alerts, reducing their responsiveness and allowing genuine model failures to go unnoticed.

ML02 Data Poisoning Attack. In the case of feedback data that is used to re-train monitoring models, altered production data can poison these models, compromising anomaly detection and feedback control systems.

- **Example of attack.** An attacker feeds maliciously crafted events into a system used to monitor malicious activity. The monitoring model is re-trained on this poisoned data and gradually learns to ignore or misclassify future malicious events.

ML08 Model Skewing. Uncontrolled or adversary-induced feedback-loop drift degrades model accuracy over time. Being gradual, it can be difficult to detect without careful monitoring of distributional changes.

- **Example of attack.** An attacker repeatedly interacts with a recommendation system in a specific way, causing it to favor similar content over time and produce biased results.

ML09 Output Integrity Attack. Flooding the inference or monitoring APIs with queries or synthetic data to exhaust observability resources, hide performance issues, or disrupt incident detection and mitigation systems.

- **Example of attack.** An attacker generates high-volume queries to overwhelm the monitoring system, causing anomalies such as performance degradation, to be obscured by noise or dropped entirely due to processing limits.

These detailed attack descriptions illustrate how adversarial actions can be embedded across various stages of the MLOps lifecycle. Understanding how each threat in its context is essential for designing effective mitigation strategies, implementing security controls, and ensuring secure AI systems.

Security measures and tools to mitigate threats in MLSecOps lifecycle stages

MLOps focuses on the streamlined and automated development, deployment, and operation of machine learning models and pipelines. Securing MLOps is critical otherwise leaving systems vulnerable to threats such as data poisoning, adversarial attacks to ML models, and weaknesses in open source libraries. MLSecOps emerges as an essential evolution of MLOps, addressing these security gaps by integrating robust security practices throughout the pipeline, establishing security as a shared responsibility among ML developers, security practitioners, and operations teams. It emphasizes securing the AI/ML starting from the planning stage, ensuring the confidentiality and integrity of model training, inference, deployment, operation, and maintaining a secure development environment.

Using MLSecOps enables early identification and mitigation of security risks, which in turn facilitates the creation of secure ML models. Figure 8 provides an integrated view of the MLSecOps framework, highlighting essential security controls and illustrating the flow of artifacts throughout the pipelines. Artifacts such as datasets, ML code, models, and deployment packages must be protected.

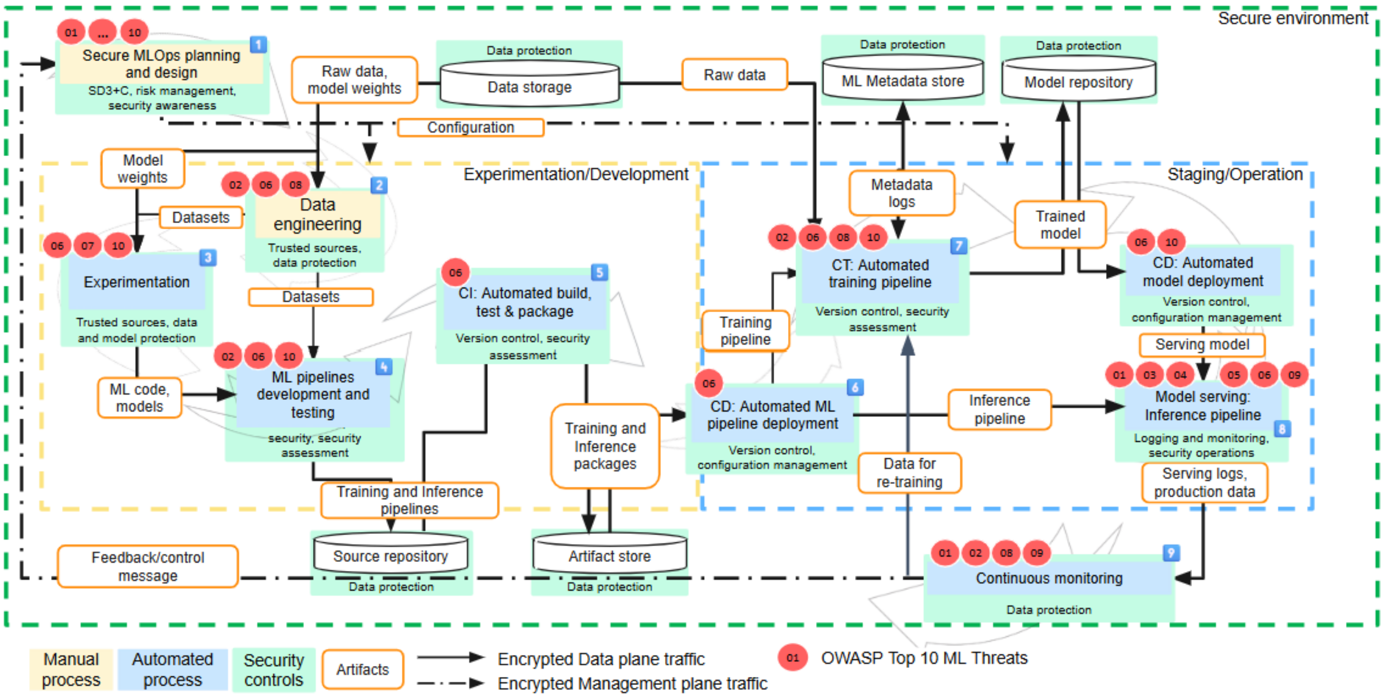
Secure environment
01 ... 10
1
Secure MLOps planning and design
SD3+C, risk management, security awareness
Raw data, model weights
Data protection
Data storage
Raw data
Data protection
ML Metadata store
Data protection
Model repository
Configuration
Experimentation/Development
Staging/Operation
Model weights
Datasets
02 06 08
2
Data engineering
Trusted sources, data protection
06 07 10
3
Experimentation
Trusted sources, data and model protection
Datasets
06
5
CI: Automated build, test & package
Version control, security assessment
ML code, models
02 06 10
4
ML pipelines development and testing
security, security assessment
Training and Inference pipelines
Training and Inference packages
Metadata logs
02 06 08 10
7
CT: Automated training pipeline
Version control, security assessment
Trained model
06 10
CD: Automated model deployment
Version control, configuration management
Training pipeline
06
6
CD: Automated ML pipeline deployment
Version control, configuration management
Inference pipeline
Data for re-training
Serving model
01 03 04 05 06 09
8
Model serving: Inference pipeline
Logging and monitoring, security operations
Serving logs, production data
Feedback/control message
Source repository
Data protection
Artifact store
Data protection
01 02 08 09
9
Continuous monitoring
Data protection
Manual process
Automated process
Security controls
Artifacts
Encrypted Data plane traffic
Encrypted Management plane traffic
01
OWASP Top 10 ML Threats

Figure 8: Integrated security controls

In Figure 8, green boxes have been layered on to represent security controls introduced at each stage of the MLOps lifecycle, highlighting measures such as data protection, version control, and secure deployment.

The large dashed green boundary represents a secure environment, serving as the foundational layer for MLSecOps and ensuring that all MLOps components operate within a trusted execution and control context.

A secure ML development environment is critical for MLSecOps. Potential security risks should be thoroughly evaluated and mitigated in accordance with the organization's Information Security Management System (ISMS) [ISO27001, Information security management systems] guidelines. In particular, the following development tool risks should be considered:

- **Secure Tools:** Secure the tools commensurate with risk. Where practical, ensure only tools from trusted and approved sources are used. Where critical, consider creating documented reviews and/or security assessments. One approach is to classify systems as high value assets, and document risks according to business impact if the risk occurs. At a minimum, counter typosquatting attacks to ensure that the intended components are being used. Additionally, use mechanisms such as HTTPS or digital signature verification to increase confidence in the tool origin.
- **Patch Management:** Establish procedures to regularly track, identify, and apply security updates and patches for all development tools.
- **Access Controls:** Enforce strict access controls to development tools based on clearly defined roles and responsibilities, following the principle of least privilege, at a minimum for the processes for checking in changes and building results.

In some cases, the organization may not have complete control over the production environment, such as when a solution is deployed on a hosted cloud platform (HCP) or a customer's private cloud. For these cases, security responsibilities must be shared between the organization and the environment provider. Furthermore, assessing the provider's security controls should be incorporated as a regular component of the overall risk assessment process.

The following identifies some example specifications and tools that can help implement a secure MLSecOps process. We do not claim that these are the best or only such examples, but instead provide them as concrete examples to show how this can be done in practice.

To effectively secure ML systems, teams must know what they are securing against. The Threats Across MLOps Stages Section provides a comprehensive taxonomy of ML-specific threats based on OWASP ML Security Top 10. It defines the scope of risk that MLSecOps processes must address and serves as a critical foundation for security planning and threat modeling.

Where OWASP Top 10 ML threats identify what to secure against, OpenSSF tools provide the means to do so, particularly for securing the ML software supply chain. For example:

- Sigstore enables cryptographic signing of ML models, to protect against model-related supply chain attacks, ensuring that artifacts are tamper-proof across deployment and retraining stages. This only works when signatures are *verified* to ensure that the signed items are from appropriate sources.
- OpenSSF Scorecard evaluates the overall health and security posture of software projects, including ML workflows, by assessing critical factors such as dependency update cadence, adherence to vulnerability management practices, and code review processes. By identifying outdated third-party components or projects with poor maintainability, the Scorecard helps mitigate risks like exploitation of insecure preprocessing code or compromised dependencies in ML pipelines. Its focus on proactive assessment ensures that security is ingrained in the development of ML systems from inception, reducing vulnerabilities that could propagate through training, deployment, or inference stages.
- Allstar enforces repository-level security controls, such as branch protection rules, required code reviews, and access management, to safeguard the integrity of ML codebases and infrastructure configurations. For instance, Allstar ensures that critical repositories hosting ML models, data pipelines, or deployment scripts cannot be altered without proper authorization or review. In the context of ML, this prevents misconfigurations or unauthorized changes that might lead to data leaks, credential exposure, or the introduction of malicious code into the pipeline. By standardizing security practices at the source, Allstar strengthens the foundational controls necessary for trustworthiness in ML operations.
- SLSA (Supply-chain Levels for Software Artifacts) introduces provenance and integrity levels that are especially applicable to ML pipelines. If model skewing or injection is suspected, SLSA-level attestations allow

teams to trace where and how a model was built, enabling root cause analysis and response.

- GUAC (Graph for Understanding Artifact Composition) can be used as a telescope to inspect model and data lineage across multiple ML pipelines. It can be used to help trace a bad prediction to the dataset from which the model learned it and also to determine what models need to be retrained once one data source reaches its end-of-life (either due to data retention regulations or due to it being found out to be malicious).

	Design	Data eng.	Exper.	Pipeline Dev	CI	CD	CT	Model serving	Sec. mon.
Sigstore		✔		✔	✔	✔	✔	✔	
OpenSSF Scorecard	✔	✔	✔	✔	✔				
Allstar	✔	✔	✔	✔	✔				
SLSA	✔	✔	✔	✔	✔	✔	✔	✔	✔
GUAC	✔	✔	✔	✔	✔	✔	✔		✔

Table 3: Illustrate how these OpenSSF tools align with key MLOps stages

In addition to tools from OpenSSF, several security tools from OWASP's suite can be effectively adapted for use within MLSecOps, even though they were originally developed for traditional web and application security:

1. Threat Dragon allows teams to model potential threats at the design stage by creating visual representations of data flows and system components. While not tailored to ML, it can be adapted to capture threat scenarios across MLOps components such as data ingestion, model training, and inference endpoints.
2. CycloneDX One of the specifications for generating Software Bills of Materials (SBOM). While it provides a mature baseline for software supply chain transparency, it does not yet cover key components on machine learning pipelines. The other widely-used SBOM specification is SPDX, whose most recent version includes mechanisms for recording AI-related information. CycloneDX and SPDX are the two most common language-independent SBOM specifications.
3. SAMM Provides a maturity model and security baselines to guide ML system design and implementation. While it lacks ML-specific extensions, it offers a structured framework for assessing and improving software security practices.
4. Dependency-Check Check identifies vulnerabilities in open source libraries, particularly relevant in ML workflows that rely heavily on external packages for preprocessing, orchestration, or visualization. Scanning these components early helps reduce exposure to known exploits and supports secure data pipeline development.
5. Threat Modeling Cheat Sheet General guidance, applicable when planning experiments that involve external data or models.
6. Dependency-Track Monitors software component risks during CI builds and alerts when vulnerabilities are discovered.

Figure 9 visualizes the integration of OpenSSF and OWASP tools across key stages of the MLOps lifecycle. Each green numbered circle is an OWASP tool that corresponds to a specific MLOps stage while the stars represent the OpenSSF tools. This layered view highlights where each tool contributes to security coverage within the lifecycle.These tools play a foundational role in traditional application security practices.

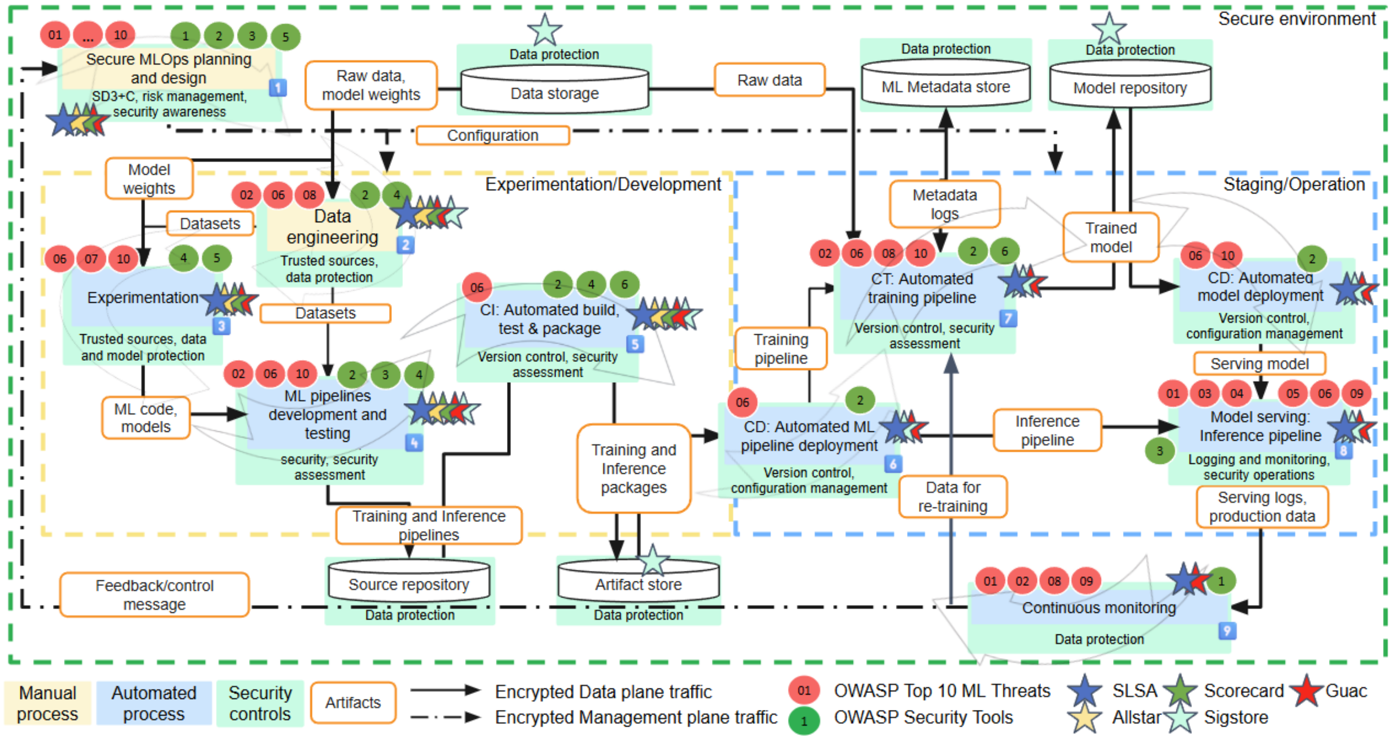
Secure environment
01 ... 10
1 2 3 5
Secure MLOps planning and design
SD3+C, risk management, security awareness
1
Raw data, model weights
Data protection
Data storage
Raw data
Data protection
ML Metadata store
Data protection
Model repository
Configuration
Model weights
Datasets
02 06 08
2 4
Data engineering
Trusted sources, data protection
2
Experimentation/Development
Staging/Operation
Metadata logs
Trained model
06 07 10
4 5
Experimentation
Trusted sources, data and model protection
3
Datasets
06
2 4 6
CI: Automated build, test & package
Version control, security assessment
5
02 06 08 10
2 6
CT: Automated training pipeline
Version control, security assessment
7
06 10
2
CD: Automated model deployment
Version control, configuration management
Serving model
ML code, models
02 06 10
2 3 4
ML pipelines development and testing
security, security assessment
4
Training pipeline
06
2
CD: Automated ML pipeline deployment
Version control, configuration management
6
Training and Inference packages
Inference pipeline
01 03 04 05 06 09
Model serving: Inference pipeline
3
Logging and monitoring, security operations
8
Data for re-training
Serving logs, production data
Training and Inference pipelines
Source repository
Data protection
Artifact store
Data protection
Feedback/control message
01 02 08 09
1
Continuous monitoring
Data protection
9
Manual process
Automated process
Security controls
Artifacts
Encrypted Data plane traffic
Encrypted Management plane traffic
01
OWASP Top 10 ML Threats
1
OWASP Security Tools
SLSA
Scorecard
Guac
Allstar
Sigstore

Figure 9: Mapping of security measures and tools to MLSecOps stages

While several of these tools, such as Dependency-Check, and Dependency-Track, are partially useful in scanning libraries or securing inference APIs, they fall short in addressing ML-specific artifacts like model weights, training data provenance, and adversarial robustness. Specifications like CycloneDX and tools like Threat Dragon, offer strong foundations for SBOM generation and threat modeling, but require targeted extensions to represent ML workflows, data pipelines, and model lifecycles accurately. Likewise, SAMM provides a valuable maturity framework, yet lacks coverage for continuous training, model retraining risks, and runtime inference security.

Our analysis underscores a key insight: while OWASP and OpenSSF tools remain relevant, MLSecOps introduces novel requirements that demand extensions to existing capabilities, particularly around model transparency, data integrity, and reproducibility, to ensure comprehensive protection of AI systems.

In addition to the OWASP and OpenSSF tools discussed, a broader landscape of security solutions exists across the open source community. These include tools for data validation, model explainability, adversarial robustness, and runtime behavior monitoring. Repositories like [awesome-MLSecOps] provide a community-curated inventory of these tools. While these tools vary in scope, many serve key protective roles within MLSecOps lifecycle.

Security measures and tools to mitigate threats in MLSecOps lifecycle stages

The following table consolidates a range of tools, some well-established and other emerging alongside the specific MLSecOps lifecycle stages they support. This view aims to guide practitioners in addressing the security concerns that arise at different stages of the ML pipeline. Some tools apply to multiple lifecycle stages, so they may repeat.

	MLSecOps Stage	Security Measures & Practices	Tools: (not comprehensive)
1	Secure MLOps Planning and Design	Threat modeling, secure design patterns	OpenSSF: Scorecard, Allstar, SLSA, GUAC OWASP: Threat Dragon, CycloneDX, SAMM, Threat Modeling Cheat Sheet Open Source Community: SPDX, Syft, Adversarial ML Threat Matrix
2	Data Engineering	Data validation, versioning, and protection. Anomaly detection, lineage tracking	OpenSSF: Sigstore (model signing), Scorecard, Allstar, SLSA, GUAC OWASP: CycloneDX, Dependency-Check Open Source Community: Deequ, Great Expectations, Data Version Control DVC, ARX Data Anonymization, YData Profiling
3	Experimentation	Supply chain security, model version control, and poisoned data detection	OpenSSF: Scorecard, Allstar, SLSA, GUAC OWASP: Dependency-Check, Threat Modeling Cheat Sheet Open Source Community: MLFlow, DVC, ART (Adversarial Robustness Toolbox), NB Defense
4	ML Pipeline Development & Testing	Reproducibility, secure artifact validation, CI testing on pipelines	OpenSSF: Sigstore (model signing), Scorecard, Allstar, SLSA, GUAC OWASP: CycloneDX, Dependency-Check, SAMM Open Source Community: MLRun, AFL++, Giskard
5	Continuous Integration (CI)	Static/Dynamic analysis, policy enforcement, dependency scanning	OpenSSF: Sigstore (model signing), Scorecard, Allstar, SLSA, GUAC OWASP: CycloneDX, Dependency-Check, Dependency-Track Open Source Community: ModelScan, Grype
6	Continuous Deployment (CD)	Secure deployment automation, model artifact checks, install packages from secure sources	OpenSSF: Sigstore (model signing), SLSA, GUAC OWASP: CycloneDX Open Source Community: Jenkins, ArgoCD, Bandit
7	Continuous Training (CT)	Continuous data validation, drift detection, model versioning, continuously authenticate feedback data	OpenSSF: Sigstore (model signing), SLSA, GUAC OWASP: CycloneDX, Dependency-Track Open Source Community: Whylogs, TensorFlow Privacy, Evidently

	MLSecOps Stage	Security Measures & Practices	Tools: (not comprehensive)
8	Model Serving (Inference Pipeline)	Input validation, access control, model watermarking, output filtering	OpenSSF: Sigstore (model signing), SLSA OWASP: SAMM Open Source Community: Garak, Seldon Core, ProtectAI/LLM-guard, TextAttack, Foolbox
9	Continuous Monitoring	Drift detection, anomaly detection, alerting, adversarial monitoring	OpenSSF: SLSA, GUAC OWASP: Threat Dragon Open Source Community: Evidently, WhyLogs

Table 4: Summary on MLSecOps stages and Top 10 threats

Table 4 illustrates how MLSecOps extends security awareness and tooling across each stage of the MLOps lifecycle, from planning and data engineering to deployment and continuous monitoring. By aligning stages with targeted practices and relevant tools, teams can proactively identify, mitigate, and respond to risks that threaten the integrity, confidentiality, and availability of ML systems. While the tools listed are not exhaustive, they represent a growing ecosystem of open source and enterprise-ready solutions that enable defense-in-depth across ML pipelines. As the field matures, organizations must continue evolving their security posture, adapting traditional software security principles to address the unique dynamics of ML-driven systems.

Secure MLOps design

Secure MLOps design involves integrating security practices into the ML lifecycle, including planning, development, deployment, and operations.

During secure MLOps design:

- Identify the key MLOps principles, components, and roles.
- Gain a comprehensive understanding of the MLOps architecture.
- Define the workflows - the sequence of tasks executed throughout the MLOps process.

Design Security Measures

Establishing a security baseline provides a foundation for the AI/ML development lifecycle. The baseline takes into account threats to AI/ML systems and includes minimum security controls, best practices, and guidelines. It is the starting point for protecting the AI/ML system and data.

Once the baseline is in place, a security risk assessment (RA) helps identify and prioritize AI/ML risks, allowing for effective risk mitigation strategies through MLOps processes.

Useful tools and references to help with risk assessment include:

- The STRIDE framework, as illustrated in "Modeling Threats to AI-ML Systems Using STRIDE," addresses vulnerabilities and specifies tools.
- "Microsoft AI Security Risk Assessment" offers an exhaustive analysis.
- MITRE's ATLAS provides data on tactics, techniques, and case studies.
- OWASP ML Security Top Ten.

- National Institute of Science and Technology (NIST) AI Risk management Framework (RMF) provides guidance for risk management.
- ISO/IEC 23894: Information technology — Artificial intelligence — Guidance on risk management.

Consider a scenario in which an anomaly detection model detects abnormal behavior and then automates solutions. Identifying risks such as data poisoning, model evasion, and Denial-of-Service (DoS) during software design enables the implementation of essential safeguards. To ensure seamless integration, these security components need to be aligned with MLOps processes. This way, they undergo development and testing concurrently with ML artifacts such as ML models and pipelines.

The secure MLOps design process also requires a secure configuration of the entire MLOps architecture, including specific processes and security controls.

Design Tools

Integrating security into the planning and design stage of the MLOps lifecycle demands not only conceptual frameworks but also practical tools that support secure architecture definition and risk assessment. Several open source tools from the OWASP and broader MLSecOps ecosystems are especially helpful during this initial stage:

- Threat Dragon: During the secure design stage, it enables teams to visually model their MLOps architectures, identify potential security threats, and document mitigation strategies early in the development lifecycle. This is particularly useful for teams new to formal threat modeling.
- CycloneDX, SPDX and Syft: By generating SBOMs, these specifications and tools provide visibility into open source packages, datasets, and model artifacts and help mitigate risks related to the AI supply chain (e.g., ML06: AI Supply Chain Attack).
- Adversarial ML Threat Matrix: A tactical guide and mapping system to help teams identify threats such as data poisoning, model theft, or membership inference — allowing their mitigation to be included at the design level.

The absence of ML-specific design tooling leaves blind spots. For example, planning might not include verifying the integrity of pre-trained models or datasets obtained from third parties, which is a supply chain gap. If a compromised model is used as a baseline (ML07: Transfer Learning Attack), a backdoor could persist into production with no mitigation planned. Similarly, lack of design for strong access control around model artifacts could later enable model theft or tampering. Essentially, without an upfront security architecture considering the OWASP ML Top 10, downstream controls may be reactive or insufficient. While the design stage benefits from a strong foundation of open source tools provided by OWASP, OpenSSF, and other open source contributors, more ML-specific and automated solutions are needed. The open source community has an opportunity to extend existing tooling or develop new ones that align security-by-design with the unique architectural components of ML systems. Investments in this space will significantly strengthen the early stages of MLSecOps adoption.

Data Engineering lifecycle stage

Data engineering takes raw data as input and produces datasets needed by subsequent processes. Security policies and controls must be enforced for data acquisition, validation, and storage, especially when aggregating data from diverse sources.

- Collected data might include sensitive personal details. The appropriate legal and contractual authority must be in place for processing sensitive data.
- In practice, data pipelines may ingest content from multiple providers or web crawls, each varying in reliability, coverage, and risk. Therefore, trust levels should be explicitly assigned to data buckets, and access controls should reflect their sensitivity and validation status. Where applicable, data acquired from lower trust levels must be vetted through the appropriate line of escalation. As larger and a

variety of data sets are used, applying this principle is challenging, and provides a space for future data security innovation. Implementation will rely on the risk of the system on sensitive data sets considered high risk use cases for each organization.

Unwanted or malicious information in the data could impact performance or introduce malicious behavior, as can tampering with stored data. To avoid this and maintain privacy, data must be properly secured at rest.

Data Engineering Security Measures

The following security measures should be implemented:

- Data storage should employ security controls appropriate for the sensitivity of the data.
- If the sensitivity demands encryption, use strong encryption algorithms.
- Stored data should be integrity-protected.
- Use separate access control for buckets of differing trustworthiness and ensure data cleaning jobs do not inadvertently cross contaminate high- and low-trust datasets.
- Data access should be monitored and logged using a formal access control process.
- Implement versioning and formal change control processes.
- Data processing pipelines should never delete/replace the original data, so that the pipelines and datasets can be independently improved.
- Label data sources with assigned trust levels and track lineage across pipelines.
- Conduct regular vulnerability scans to identify potential poisoning threats using updated tools.
- Perform regular data backups and recovery tests.
- Implement data retention policies on storage lifetime and secure disposal.

Data quality strongly influences model quality, so data quality should be maximized throughout the development lifecycle. Similarly, data protection measures must be implemented during development and production. During development, acquired training data should be continuously assessed for trustworthiness, anomalies, or hidden manipulations. In certain cases such as anomaly detection, it can be challenging as anomalies can be real-world events or could be an outcome of a malicious attack. Therefore, to analyze suspicious data effectively, the involvement of subject-matter experts is crucial.

Data Engineering Tools

To support secure data engineering practices within the MLOps lifecycle, a number of tools from the open source community can help enforce controls over data integrity, privacy, access, and provenance.

- Great Expectations and Deequ: Two widely used tools for validating and profiling datasets. They help ensure data quality at ingestion time, detect schema anomalies, and enforce business rules, forming the first line of defense against poisoned or malformed data.
- DVC (Data Version Control): Supports robust dataset versioning, enabling traceability and rollback capabilities. It can be integrated with access-controlled storage and CI pipelines, aligning with formal change control and secure retention policies.
- ARX Data Anonymization Tool: Relevant where privacy-enhancing techniques are needed. The tool enables structured data anonymization and image/video redaction, respectively, essential when handling sensitive datasets.
- YData Profiling: Provides automated exploratory data analysis with visual profiling reports that are useful for identifying missing data, outliers, and potential red flags before integration into production pipelines.

Experimentation

While conducting experiments, a data scientist performs ML model engineering, selects features, and algorithms or develops new ones, trains the model, and tunes hyperparameters. The inputs consist of model weights and datasets, and the outputs include ML code and ML models.

Incorrect or insecure code can result in availability, integrity, or confidentiality risks. Security practitioners should consider the following requirements:

- Conduct design and research in a secure environment
- Review and approve model selection at an early stage in development before implementing it in production. Track the model throughout
- Document experiments and associated Implement full traceability of experiments and model training

An ML model trained under ideal conditions might prove fragile when deployed in potentially adversarial environments. Metrics and testing sets should emulate different kinds of drift and anticipated adversarial conditions.

Experimentation Security Measures

The following measures should be taken:

- Ensure that training, validation, and testing sets adhere to natural temporal dependencies
- Enhance model robustness by augmenting datasets with common corruptions that could reasonably be encountered
- If adversarial examples are a concern, consider adversarial training
- If training uses distributed data, consider federated learning to mitigate privacy concerns

Version control and integrity protection should be used to detect unauthorized changes in the ML model, which helps detect poisoning. Signed and integrity-protected versions enable reversion to a known good state in the event of corruption.

When transferring ML models, it is important to safeguard them from unauthorized alterations. A standard approach is to apply a cryptographic hash function over the model. Hashes should be encrypted or transmitted on alternate channels.

Trained models are intellectual property that should be protected, according to the desired transparency, security risk assessment, and the relative value of the model. Training scripts and feature engineering codes could have even higher intellectual property value and merit protection. Model, training, feature calculation can be protected by confidential computing, encryption, and obfuscation.

ML model validation procedures should include comprehensive security testing. Regular testing can detect compromised ML models through simple checks, while advanced testing uses a broader range of attacks to identify vulnerabilities. Additionally, custom and threat-based scenarios should be developed for penetration testing purposes.

Experimentation Tools

Security during experimentation can be enhanced by integrating tools that support reproducibility, traceability, robustness testing, and adversarial defense. The following open source tools can assist teams in applying secure practices during this stage:

- MLflow: One of the most widely used platforms for managing the machine learning lifecycle. It supports experiment tracking, model versioning, and artifact logging, which together enable full traceability of training activities and model lineage. This traceability is critical for auditability and detecting tampering with model artifacts.
- DVC: Adds Git-like version control for data, models, and ML pipelines. It helps detect unauthorized changes to datasets or training scripts.
- Adversarial Robustness Toolbox (ART): Provides a suite of attacks and defenses for testing ML model robustness. It allows security practitioners to evaluate the vulnerability of models to evasion, poisoning, and inference attacks and supports adversarial training for hardened defenses.

- Model Transparency (via Sigstore): Enables digital signing and verification of machine learning models to ensure integrity and authenticity. This tool allows teams to cryptographically sign models at critical development stages (e.g., training, deployment, retraining), and protect against tampering or supply chain attacks.

ML pipeline development and testing

Automated training and inference pipelines are used for continuous training and model serving. The process takes datasets, ML codes, and models from the experimentation stage as inputs and outputs in training and inference pipelines. Creating ML pipelines should follow a software development lifecycle (SDLC) like any other software development process.

Pipeline security includes version control, integrity, and confidentiality protection. Pipeline protection is like ML model protection. Both require attention to confidentiality, integrity, access controls, and full lifecycle compliance with established policies and regulations.

The security considerations for ML code and parameters used in ML pipelines should be defined and agreed upon. Security measures should be implemented during pipeline development, security testing practices should be aligned for application in pipeline testing.

Developing and Testing Security Measures

Consider the following security practices for pipeline development:

- The Software Assurance Maturity Model (SAMM) by OWASP provides a framework for incorporating security activities into software development and maintenance.
- Code review or peer reviews, including those conducted by a software engineer.
- Static Application Security Testing (SAST) examines software security without execution by analyzing either the source or the compiled binary.
- Dynamic Application Security Testing (DAST) assesses the software security in a runtime environment through execution.
- Fuzz testing provides invalid input (randomly generated or specifically crafted) to an execution and monitors a pipeline for crashes, buffer overflows, or other unexpected results.
- Interface (API) testing involves multiple teams working on different parts of an ML pipeline.
- Misuse or abuse case testing simulates user attempts at manipulating inputs to produce a corrupted ML model.

During continuous training, an automated training pipeline must produce models that behave similarly to those created during experimentation, provided the same inputs are used. Similarly, the inference pipeline should produce results that align with those achieved during experimentation. A reference model, derived from the experimentation stage, should be integrity-protected and signed.

Developing and Testing Tools

Securing ML pipelines requires an approach similar to traditional software engineering with a variety of open source tools that can be leveraged during pipeline development and testing.

- MLRun: A robust MLOps orchestration framework that supports building and testing ML pipelines with versioning, reproducibility, and auditability in mind. It enables model training and inference workflows to be packaged, logged, and validated in a standardized manner.
- OWASP Dependency-Check, SPDX, and OWASP CycloneDX: These tools llow pipeline developers to scan libraries and components used in pipeline stages for known vulnerabilities. These tools also generate

SBOMs for auditing ML pipelines that rely on multiple open source dependencies.

- AFL++: A fuzzing tool that can be adapted to ML services within pipelines to stress-test APIs and intermediate stages using malformed inputs, exposing runtime vulnerabilities that could corrupt models or outputs.
- Giskard: Helps implement automated testing of ML models, including robustness, bias, and privacy leakage tests. When integrated into pipelines, it enables regression testing of model behaviour across continuous retraining cycles.

Together, these tools enable development teams to embed security across the ML pipeline lifecycle, from static testing of pipeline code to dynamic validation of runtime components. However, one notable gap is that static code scanners do not understand ML context. They might catch a generic issue (e.g., use of eval() or a dangerous file permission setting) but will not warn if a model training code lacks input normalization or if an evaluation script is insufficient. Also the tools cannot catch errors like an insecure model serialization method, but a typical SAST rule-set might not flag such weakness in an ML context. Addressing this would further bridge the gap between MLOps and traditional application security practices.

Continuous Integration/ Continuous Delivery and Deployment, Continuous Training

In Continuous Integration/Continuous Delivery and Deployment (CI/CD), and Continuous Training (CT) stages pipelines, model artifacts, and other relevant assets are often transmitted between environments, and should be protected from modification in transit.

When an ML model is embedded in a solution or product, a separate verification for the model's authenticity might not be necessary if the authenticity of the solution inherently validates the model. However, if the ML model is supplied independently, such as during a version update, the authenticity of the model must be verified. The following general aspects of securing CI/CD should be investigated:

- Encrypt and integrity protect artifacts in transit and at rest.
- The target environment should be able to perform authenticity checks of signed artifacts.
- Use version control to prevent updating with an old, potentially vulnerable ML model.

CI Security Measures

For CI, the inputs are training and inference pipelines, while the outputs are training and inference packages. Key concerns include insecure code, insecure or outdated third- party dependencies (vulnerable to known attacks), build artifacts containing sensitive information, and insecure configurations. The following measures can be considered for CI:

- The build environment should be isolated and securely configured.
- Third-party dependencies should be scanned for vulnerabilities and updated promptly.
- Securely store and transmit build artifacts.

CD Security Measures

While CD accelerates ML pipeline delivery or deployment, it can introduce security concerns if not managed properly. For automated ML pipeline deployment, the inputs include training and inference packages, and the outputs are the deployed pipelines. For automated model deployment, the inputs are trained ML models and the outputs are serving ML models. The CD pipelines should guarantee the secure delivery and deployment of an ML pipeline or the serving model. The following security concerns require attention:

- Input ML pipelines should be validated, integrity-protected, and signed by the person responsible for their security testing.

- The CD pipeline, delivery, and deployment environment should be securely configured and evaluated regularly.
- The CD pipeline should track the delivery and deployment artifacts and block sensitive ones. Relevant logs should be confidentiality and integrity protected. Delete unnecessary artifacts.

CT Security Measures

CT refers to regularly retraining an ML model by incorporating new data. It is facilitated by a monitoring component, a feedback loop, and an automated training pipeline that takes the raw data and executes the necessary preprocessing and training steps. CT takes raw data and a training pipeline as inputs to produce a trained model. Metadata logs serve as both input and output artifacts and should be secured accordingly. Model evaluation and validation are critical components of CT, as they analyze any changes in model quality and security. Security considerations for CT include:

- Regular security assessments and patching to keep the training pipeline secure
- Data integrity checks to prevent data tampering or injection attacks
- Model evaluation, which assesses changes in model quality, must be conducted in a secure environment

CI/CD/CT Tools

Ensuring the security of CI/CD and Continuous Training (CT) workflows requires enforcing strong guarantees around artifact authenticity, pipeline isolation, dependency hygiene, and secure metadata management. Several open source tools from the OWASP and MLOps communities can help enforce these practices throughout the delivery and retraining lifecycle.

- OWASP Dependency-Track SPDX, and OWASP CycloneDX In the context of ML CI/CD, these tools help validate ML dependencies before they are embedded into training or inference environments.
- Argo CD is a declarative GitOps continuous delivery tool for Kubernetes environments. It helps secure the deployment pipeline by maintaining a version-controlled source of truth and enforcing policy-based rollouts, while tracking configuration drift and ensuring reproducibility.
- Evidently and WhyLogs are valuable in CT pipelines, enabling model validation and quality monitoring. Their outputs feed into retraining triggers and help determine if changes in data warrant retraining or raise concerns of data poisoning.
- Sigstore Provides a set of tools and infrastructure that allows developers to sign and verify software artifacts, including ML models and associated metadata, using cryptographic signatures.

Model Serving

Model serving is the process when a trained ML model makes inferences in a production environment. Before the model would serve client requests, it must be deployed to the MLOps system. This stage introduces a new set of security challenges that require safeguarding both the model and the inference service from misuse, reverse engineering, and adversarial manipulation. The serving ML model and the inference pipeline must be properly secured to ensure safe and reliable operation. Note that in some cases an ML model is served to an external client and thus executed outside the control of the organization delivering the ML model.

Model Serving Security Measures

Secure implementation, configuration, and testing of the serving process include:

- A properly configured container and orchestration environment
- Robust access control mechanisms for the inference service.
- Data encryption and access privacy-enhancing technologies or data de-identification techniques (such as anonymization or pseudonymization) might be employed.

- Model encryption, watermarking, or homomorphic encryption to protect against reverse-engineering, and intellectual property (IP) or sensitive data leakage.
- Model inversion and membership inference protections. Model usage should be monitored. When handling direct client requests rather than consuming data streams or processing batches from a single source, it is important to restrict the number of client requests.
- Evasion/adversarial attack protections. The model should be hardened against malicious attacks. Use input validation and sanitization algorithms, limit the number of queries, and implement proper adversarial robustness techniques.

Additionally, regular security audits, least privilege policies, and scalability in security measures can significantly enhance the robustness of the serving process.

Model Serving Tools

Several open source tools and libraries, originating from MLOps and MLSecOps can support secure serving of ML models.

- Garak: A red-teaming tool for evaluating model vulnerabilities against prompt injection, input manipulation, and output leakage. It is particularly relevant for serving LLMs and can be used in security testing before and after deployment.
- Seldon Core: Built for deploying and orchestrating machine learning models on Kubernetes infrastructure. It simplifies the transition from development to production by supporting models from diverse frameworks—such as TensorFlow, PyTorch, and scikit-learn—and wrapping them as scalable, containerized services.
- TextAttack and Foolbox: Can be used to evaluate serving models for vulnerability to evasion attacks. While often associated with pre-deployment testing, these tools are also valuable in configuring and validating defensive mechanisms for runtime inference services.

Security Monitoring

Securing AI/ML systems is a continuous process that extends beyond development and deployment. Operational procedures to create a controlled and secure environment should be standardized.

Security Monitoring Security Measures

Introduce activity monitoring by implementing actionable dashboards, displaying critical metrics such as:

- Model performance indicators
- Usage statistics
- Metadata related to inference requests
- Input and output error tracking

Events can be sorted by significance, facilitating prioritized investigation. Identifiable error types can help categorize and correlate events. This aids in identifying patterns and trends that might not be apparent when looking at events in isolation.

Automated detection and response mechanisms are important. Drift monitoring helps maintain detection capabilities, by helping to detect malicious input, such as adversarial examples. AI/ML-specific monitoring systems should be integrated into the overall dashboard. Simple alerts are effective for regular performance incidents, but advanced security filters and custom responses might be necessary for unexpected inputs or crashes.

Regardless of the specific response, the AI/ML system's issues must be promptly addressed, and best practices and processes continually updated. This includes providing additional training to the relevant personnel.

Security Monitoring Tools

Security monitoring of AI/ML systems requires the integration of both general observability frameworks and ML-specific instrumentation. These tools should facilitate real-time detection of performance degradation, adversarial activity, and abnormal usage patterns. A combination of open source tools from the MLSecOps and observability domains can be used to establish a robust monitoring foundation.

- Evidently: Provides capabilities for monitoring model quality, data drift, target drift, and feature importance in real time. It supports dashboard integration and alerting, making it suitable for both operational and security monitoring use cases.
- WhyLogs: A lightweight, scalable logging system tailored for ML applications. It logs statistical profiles of datasets and model outputs, supporting anomaly detection and helping to flag potentially malicious input behavior in production pipelines.

Challenges and recommendations

Challenges

Introducing MLSecOps represents a significant evolution in securing AI and ML lifecycles. However, organizations embarking on this journey often encounter substantial challenges arising from the inherent complexity, dynamic nature, and opacity of AI/ML technologies. Addressing these issues demands specialized competencies, strategic approaches, and a thoughtful integration of security practices explicitly tailored to AI/ML contexts. Additionally, many technologies in this field are still emerging and relatively immature, lacking comprehensive security measures and thus necessitating more rigorous assessment. Understanding these challenges and implementing targeted recommendations is essential for organizations that want to build and use resilient, secure AI/ML systems.

Challenge 1: Distinct Threats for DevSecOps vs MLSecOps

Machine learning systems introduce a threat landscape fundamentally different from traditional software. While DevSecOps addresses well-known risks such as injection flaws, insecure configurations, and software vulnerabilities, MLSecOps must contend with threats specific to the ML paradigm. These include data poisoning, where malicious training inputs distort model behavior; adversarial inputs that exploit model sensitivity to imperceptible perturbations; and model theft or tampering, which targets the model itself as a high-value asset. Privacy-focused attacks, such as model inversion or membership inference, leverage the model's learned parameters to reveal sensitive training data. Additionally, deployed models are exposed to misuse through APIs, with risks ranging from denial-of-service to integrity loss through unauthorized updates. These attack vectors do not map cleanly to conventional software risks, and defending against them requires security controls explicitly designed for the ML lifecycle. In many cases there are no certain defenses, merely probabilistic defenses, making it difficult to defend against any adversary who can apply many attempts.

Challenge 2: Complexity of Continuous Training

Unlike traditional software systems, machine learning workflows are iterative, data-driven, and remain operationally dynamic even post-deployment. Continuous training introduces unique complexity, where each retraining cycle can modify model behavior and expose the system to new security risks. ML pipelines often span multiple interconnected components, data ingestion, feature engineering, training orchestration, and deployment, making end-to-end assurance difficult. This complexity, coupled with frequent updates and multi-team ownership, blurs the boundaries between development and maintenance, demanding a shift toward secure, automated, and repeatable workflows. To maintain integrity and trust in ML systems, organizations must embed security controls across the entire lifecycle, treating every training iteration as a potential point of vulnerability.

Challenge 3: Managing Opacity and Interpretability in ML Models

Unlike traditional software systems where logic is transparent and traceable, most machine learning models, especially those based on deep learning, operate as opaque systems with limited human interpretability. Their complex internal representations make it difficult to understand, audit, or explain why a specific decision was made. This lack of transparency poses significant challenges for security, as malicious behavior embedded within a model may go undetected. Without clear insight into a model's reasoning, security teams must rely on indirect validation methods such as adversarial testing or explainability tools. However, current explainability techniques remain limited in their ability to reveal hidden backdoors or subtle manipulations. In safety-critical or regulated environments, this opacity further complicates certification and risk assessment. As a result, improving interpretability is not only a matter of transparency, it is a foundational requirement for establishing trust, enabling effective threat detection, and supporting secure deployment of ML systems.

One method to help reduce this risk, in a safety critical environment, is to evaluate the model in a trusted enclave, producing an attestation that commits to the (model, dataset, evaluation score) tuple where the evaluation was completed in a tamper-proof environment. Tooling in the control plane can then interpret this data and allow only models into production environments that would score higher than a specified threshold on the dataset (assuming the dataset can cover the majority of input scenarios). Outside of this method, it is possible for a human to lie or mistakenly say that a specific model performs better than actual on the test dataset (or the dataset could leak if humans are allowed to test on it directly). If evaluation is performed on a Trusted Execution Environment (TEE), the score is given in a tamper-proof attestation, the dataset is not accessible outside of the TEE. It does not make the model less opaque, but it helps in reducing the risk.

Challenge 4: Security Risks Introduced by Frequent Retraining

In contrast to static software releases, machine learning systems are frequently retrained to adapt to evolving data and operational conditions. While this enables responsiveness, it introduces security risks with each iteration. Retraining cycles may ingest manipulated data, increasing exposure to slow poisoning attacks that can subtly degrade model integrity over time. The rapid pace of updates also limits the feasibility of manual security reviews, requiring automated validation, model versioning, and rollback mechanisms. Without lineage tracking and robust evaluation protocols, retraining can propagate vulnerabilities into production. Frequent updates thus demand a continuous security posture, integrating automated checks into the training pipeline to ensure each model iteration meets integrity, performance, and resilience baselines.

Challenge 5: Challenges in Model Provenance and Reproducibility

Machine learning systems evolve rapidly through frequent retraining and updates of large-scale data sets though complex pipelines. As a result it is challenging to track every detail of the model's lineage. When models are frequently re-trained or updated, implementing effective model versioning that can track changes over time, including the parameters, architecture, and training data used for each version can be challenging. Some later training data may be the result of previous executions of the model, causing a drift that can be hard to correct.

Reproducibility presents a parallel challenge: Being able to reproduce model behavior is important for debugging and improving performance and security measures. However, the dynamic nature of models and development environments maintaining reproducibility over time becomes challenging. In practice, such pipelines are unlikely to produce the same model, and repeated inferences using the same model often produce somewhat different results. It is crucial to build policies and practices, e.g., how to take a snapshot of the configurations, training data, build environment and so on as a reference point. ML projects depend on numerous libraries and frameworks, which may change over time. Flexible tests that verify "approximate" reproduction will often be necessary. Managing these dependencies and ensuring that the same versions are used is crucial for reproducibility.

Challenge 6: Difficulties in performing Risk Assessment

Risk management for AI is challenging primarily due to difficulties in measurement, trade-offs between metrics, prioritization, organizational challenges, and defining acceptable risk tolerance. Defining the metrics to measure, spanning from security, safety, ethics, associated with AI systems requires understanding their behavior, impact, and the context in which they operate, all of which can be complex and unpredictable. There exist trade-offs between metrics which increase the complexity due to conflicting objectives. Balancing metrics, such as optimizing for accuracy versus fairness, security versus explainability, or transparency versus efficiency, requires careful consideration and prioritization.

Recommendations

To address the above challenges, organizations should take practical steps to mature their Machine Learning Security Operations (MLSecOps). Here are actionable recommendations:

- **Adopt Structured Security Maturity Models**
  Use established frameworks such as OWASP SAMM, or NIST AI RMF to baseline your current security posture and identify gaps across the ML lifecycle. Embed “security by design” into AI projects by establishing policies for secure data sourcing, model validation, and decommissioning practices.
- **Automate Security Controls Across the ML Pipeline**
  Integrate automated checks for data validation, model robustness, and drift detection into CI/CD/CT pipelines. Use tools for scanning model artifacts, verifying signatures, and monitoring ML-specific performance metrics to reduce manual oversight and detect issues early.
- **Enforce Reproducibility and Version Integrity**
  Maintain comprehensive version control for datasets, models, and training environments. Use model registries, signed reference models, and reproducible containerized workflows to ensure traceability and integrity across retraining cycles and deployments.
- **Existing security posture**
  Evaluate the AI/ML security practices during development and operation, supplemented by other security protocols, throughout the lifecycle.
- **Team collaboration**
  Promote robust cross-departmental collaboration, including the engagement of a security practitioner into the MLOps team without impeding the development process. Educate team members on the novelty and specificity of AI/ML threats and countermeasures.
- **IT Infrastructure assessment**
  Review the IT infrastructure for its compatibility with MLSecOps, which favors a flexible infrastructure over traditional rigid ones. It is important to note that MLSecOps is not a standalone solution, but rather an additional layer on top of existing cybersecurity mechanisms, which should be robust and mature. Establishing an Information Security Management System (ISMS) can facilitate MLSecOps.
- **Setting clear objectives**
  This involves automating threat detection for AI/ML, developing defense evasion susceptibility, or predicting security incidents. Specific objectives will guide MLSecOps’ strategy and help measure its effectiveness.

Steps to implement MLSecOps

- **AI/ML Security competence**
  Assign security practitioners with appropriate skills to assist the MLOps team and ensure adherence to security practices throughout the AI/ ML development lifecycle. They will also promote security awareness and help all team members understand their roles in securing AI/ML.
- **Integration of security tools**
  Automate security tooling within the development process. Integrate tools for data analysis, static code analysis, dynamic testing, and dependency checking into the CI/CD pipeline.
- **Continuous monitoring and improvement**
  MLSecOps requires regular reviews and improvements. The MLOps process should be monitored, logged, and audited regularly as new threats appear frequently. Security incident handling is valuable for learning about them and enhancing the overall MLSecOps.

Potential obstacles and ways to overcome them

- **Lack of skilled personnel**
  MLSecOps calls for a solid understanding of both ML and cybersecurity, including which threats can be mitigated with conventional controls, and which require ML-specific ones. If your organization lacks

such expertise, consider recruiting specialists or investing in competence development.

- **Data privacy concerns**
  Mitigate privacy concerns by anonymizing data, using differential privacy, and keeping up to date with the latest regulatory advancements.
- **Constantly evolving threats**
  AI/ML threats constantly change, potentially making even recently developed ML models susceptible to attacks. The model development and deployment process need to be flexible and undergo regular updates.
- **Tooling challenges**
  Integrating the appropriate security tools can be intimidating. Initially, focus on identifying the most crucial ones that can be easily integrated. As the process becomes stable, use more sophisticated tools.
- **Process overhead leading to resistance to change**
  Implementing security measures can initially slow down development, leading to resistance. Demonstrate that incorporating security in the early stages of ML preparation will minimize future concerns. Make it clear that AI/ML systems are often opaque and data-intensive. Without a comprehensive approach to protecting the AI/ML development lifecycle, potential security and privacy issues could arise later, at a greater overall cost.

Conclusion

As machine learning becomes more deeply integrated into critical systems and business workflows, its operational backbone, MLOps, demands proactive security integration to address emerging risks. This paper introduced MLSecOps as a natural extension to the MLOps lifecycle, building on DevSecOps practices, embedding security into every stage from planning and data engineering to deployment, monitoring, and continuous training.

While existing tools like those from OWASP and OpenSSF offer a solid foundation, MLSecOps calls for both adapting current technologies and developing new ones tailored for the unique demands of ML.

We introduced a new set of personas, who represent the diverse practitioners now participating in securing ML lifecycles. These personas reflect how roles are shifting across the AI/ML landscape, and how new tools and community initiatives can support each persona in practical, accessible ways, adapting to their current workflows.

Before diving into MLSecOps, organizations must evaluate their security posture, encourage cross-departmental collaboration, assess their IT infrastructure, and set clear objectives. MLSecOps should not be seen as a replacement for existing security protocols but as a supplement that enhances every stage of AI/ML development.

MLSecOps is a continually evolving approach for integrating security into the AI/ML development process. By recognizing security as a collaborative effort, we can build AI systems that are not only performant and scalable, but also secure and reliant by design.

References

- CNCF end user technology radar provides insights into DevSecOps | CNCF
- DOD Enterprise DevSecOps Reference Design: CNCF Kubernetes
- DevSecOps Days Washington DC 2021
- About – Open Source Security Foundation
- GitHub - ossf/tac: Technical Advisory Council
- GitHub - ossf/ai-ml-security: Working Group on Artificial Intelligence and Machine Learning (AI/ML) Security
- Secure Software Development Education 2024 Survey
- ML4Devs: MLOps Machine Learning LIfe Cycle
- MLSecOps: Protecting AI/ML Lifecycle in telecom - Ericsson
- toolbelt/personas/README.md at main · ossf/toolbelt · GitHub
- Sachiko the Solutions Architect
- Alison the AIML Engineer
- Timmy the Test Engineer
- Guinevere the Security Governance Lead
- Pang the Product Security Engineer
- Chinmay the Cloud Platform Admin
- Ophelia the IT Infrastructure Engineer
- Daniel the Data Scientist
- Dibby the Data Engineer
- Grear the Data Governance Analyst
- Machine Learning Operations (MLOps): Overview, Definition, and Architecture
- OWASP Machine Learning Security Top Ten | OWASP Foundation
- Sigstore
- sigstore/model-transparency: Supply chain security for ML
- ossf/scorecard: OpenSSF Scorecard - Security health metrics for Open Source
- ossf/allstar: GitHub App to set and enforce security policies
- SLSA • Supply-chain Levels for Software Artifacts
- guacsec/guac: GUAC aggregates software security metadata into a high fidelity graph database
- OWASP/www-project-threat-dragon: OWASP Foundation Threat Dragon Project Web Repository
- CycloneDX/specification: OWASP CycloneDX is a full-stack Bill of Materials (BOM) standard that provides advanced supply chain capabilities for cyber risk reduction. SBOM, SaaSBOM, HBOM, AI/ML-BOM, CBOM, OBOM, MBOM, VDR, and VEX
- OWASP SAMM | OWASP Foundation
- OWASP Dependency-Check | OWASP Foundation
- Threat Modeling - OWASP Cheat Sheet Series
- OWASP Dependency-Track | OWASP Foundation
- RiccardoBiosas/awesome-MLSecOps: A curated list of MLSecOps tools, articles and other resources on security applied to Machine Learning and MLOps systems

References

- SPDX – Linux Foundation Projects Site
- anchore/syft: CLI tool and library for generating a Software Bill of Materials from container images and filesystems
- awslabs/deequ: Deequ is a library built on top of Apache Spark for defining "unit tests for data", which measure data quality in large datasets
- Great Expectations: have confidence in your data, no matter what ▪ Great Expectations
- Data Version Control · DVC
- ARX – Data Anonymization Tool – A comprehensive software for privacy-preserving microdata publishing
- ydataai/ydata-profiling: 1 Line of code data quality profiling & exploratory data analysis for Pandas and Spark DataFrames
- MLflow
- Trusted-AI/adversarial-robustness-toolbox: Adversarial Robustness Toolbox (ART) - Python Library for Machine Learning Security - Evasion, Poisoning, Extraction, Inference - Red and Blue Teams
- protectai/nbdefense: Secure Jupyter Notebooks and Experimentation Environment
- Open Source MLOps Orchestration | MLRun
- AFLplusplus/AFLplusplus: The fuzzer afl++ is afl with community patches, qemu 5.1 upgrade, collision-free coverage, enhanced laf-intel & redqueen, AFLfast++ power schedules, MOpt mutators, unicorn_mode, and a lot more!
- protectai/modelscan: Protection against Model Serialization Attacks
- anchore/grype: A vulnerability scanner for container images and filesystems
- Jenkins
- Argo CD - Declarative GitOps CD for Kubernetes
- whylogs: the open standard for data logging | WhyLabs
- tensorflow/privacy: Library for training machine learning models with privacy for training data
- evidentlyai/evidently: Evidently is an open source ML and LLM observability framework. Evaluate, test, and monitor any AI-powered system or data pipeline. From tabular data to Gen AI. 100+ metrics
- NVIDIA/garak: the LLM vulnerability scanner
- SeldonIO/seldon-core: An MLOps framework to package, deploy, monitor and manage thousands of production machine learning models
- protectai/llm-guard: The Security Toolkit for LLM Interactions
- QData/TextAttack: TextAttack is a Python framework for adversarial attacks, data augmentation, and model training in NLP https://textattack.readthedocs.io/en/master/
- bethgelab/foolbox: A Python toolbox to create adversarial examples that fool neural networks in PyTorch, TensorFlow, and JAX
- Modeling Threats to AI-ML Systems Using STRIDE
- AI Risk Assessment for ML Engineers | Microsoft Learn
- MITRE ATLAS™
- AI Risk Management Framework | NIST
- ISO/IEC 23894:2023 - AI — Guidance on risk management

Authors

We gratefully acknowledge Ericsson for donating the **original paper** that served as the foundation for this work. This version has since been extended with significant new contributions from the original authors, Andrey Shorov and Elif Ustundag Soykan, along with major contributions from Dell Technologies security researchers Sarah Evans and Bahaulddin Shammary.

The OpenSSF AI/ML Working Group also extends our appreciation to the reviewers for their valuable feedback, which helped improve the quality and clarity of the paper: Eddie Knight, Christopher Robinson, David A. Wheeler, Mihai Maruseac, Rob Moffat, and other OpenSSF and AI/ML WG members.

OpenSSF
OPEN SOURCE SECURITY FOUNDATION

Thank you! Join us..

openssf.org/getinvolved