Domain 8: Cloud Workload Security

Introduction

A cloud workload is a related set of software and data units that are deployable on some type of infrastructure or platform. Examples include software packages that run in a virtual machine (VM), containers, and Functions as a Service (FaaS). In this domain, we both discuss those units and the platforms on which they are deployed. This aligns with a discussion on the risks specific to the workload, and on how responsibility for treating those risks is shared between cloud consumer and provider.

Learning Objectives

In this domain, you will learn to:

- Understand the challenges and uniqueness of creating security approaches for cloud security workloads.
- Understand security considerations for virtual machines.
- Understand security considerations used for securing containers.
- Understand security considerations to provide PaaS security.
- Understand security considerations for securing serverless or function as a service workloads.
- Understand security considerations for AI Workloads.

8.1 Introduction to Cloud Workload Security

Key differences in cloud workloads include:

- **Dynamic and Expansive:** Cloud environments are constantly evolving, requiring agile and adaptable security approaches.
- **Complexity and Diversity:** Various workload types each have unique requirements, making a one-size-fits-all security approach ineffective.
- **New types of workloads:** serverless, functions as a service, and managed containers are workloads that are popular in cloud environments.

8.1.1 Types of Cloud Workloads

Cloud workloads vary, each with distinct security implications. Here are some examples:

- **Virtual Machines (VMs):** Offer isolation through separate operation systems (OS) and hypervisor-managed boundaries. The security of guest OS is the CSC's responsibility.
- **Containers:** Share host OS kernel, offering weaker isolation than VMs. Security relies on OS-level controls and container image management.
- **FaaS:** Functions run without infrastructure management. The CSP handles most security, but the CSC must manage secrets and permissions.
- **AI Workloads:** Process large data sets, presenting unique security challenges.

Management responsibilities often shift to CSP, but visibility and control challenges remain.

8.1.2 Impact on Workload Security Controls

Cloud workload security involves setting up and maintaining effective technical controls. Traditionally, these were often implemented in the network or manually installed. Key considerations include:

- **Enforcing Controls:** Use lightweight, cloud-aware security agents like Endpoint Protection Platforms (EPP) or Endpoint Detection and Response (EDR). Avoid inbound network ports to minimize attack surfaces.
- **Monitoring:** Capture workload logs and send them to centralized storage promptly. Enrich log entries to account for dynamic workload identities.
- **Assessment:** Conduct vulnerability scans on VM and container images before deployment, perform offline runtime assessments, and build assessment agents into images.
- **Cloud Workload Protection Platforms (CWPP):** Utilize platforms tailored for cloud workloads to provide comprehensive security capabilities, including vulnerability scans, log collection, and runtime protection.

8.2 Securing Virtual Machines

Virtual Machines (VMs) are full operating systems running on hypervisors, commonly used for cloud workloads due to their proximity to hardware and widespread familiarity. They ensure strict isolation between workloads within and across customers. Each VM operates with its own complete operating system stack.

VM deployment starts from a standardized base image, ensuring consistent security configurations. The cloud's autoscaling feature enables the use of immutable workloads, improving efficiency and adapting to changing demand.

8.2.1 Virtual Machine Challenges & Mitigations

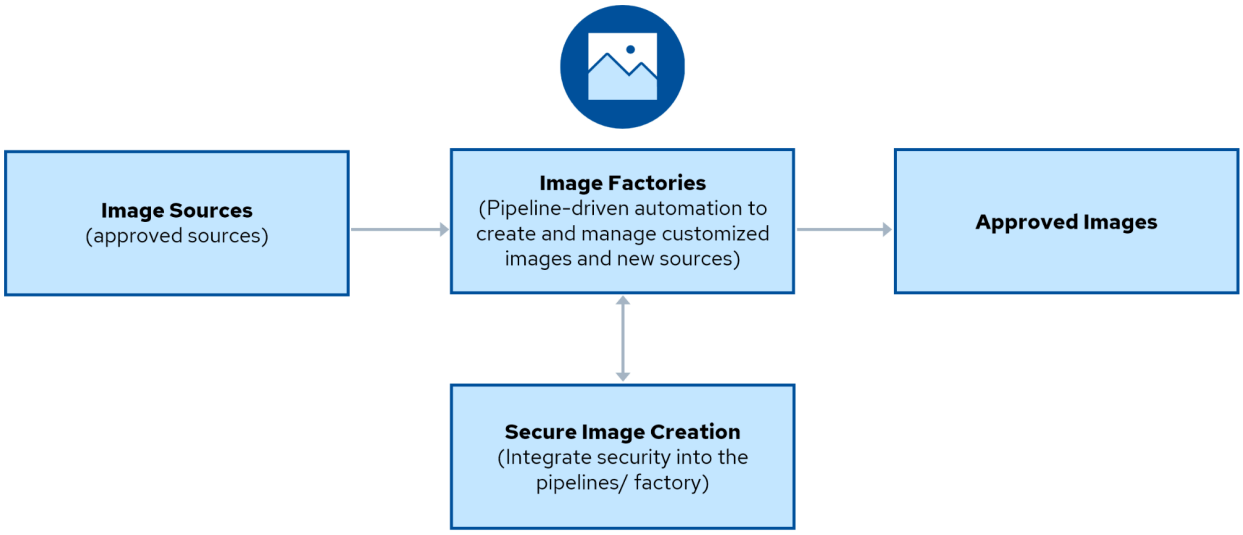
Securing VMs involves addressing unique challenges and implementing best practices:

- **Use secure base images:** Enforce from a managed catalog, versioned and immutable
- **Patch management:** Regularly update images with security patches
- **Minimize attack surface:** Remove unnecessary OS components and harden configurations
- **Access controls:** Implement least privilege principles
- **Automation:** Utilize for scanning, patching, and reporting
- **Configuration management:** Use Infrastructure as Code (IaC)
- **Monitoring and logging:** Centralize and track activities
- **Network security:** Harden Secure Shell (SSH) and use host-based firewalls
- **Secure boot:** Protect against pre-boot malware attacks
- **Specialized security tools:** Monitor hypervisors continuously

Efficient management ensures robust security and adaptability for cloud workloads.

8.2.2 Creating Secure VM Images with Factories

Creating and managing VM images is crucial for securing VM environments. It involves embedding security measures from the start, requiring streamlined practices to integrate security seamlessly at every layer. Key aspects include image factories and image sources.



Approved sources + approved process = approved images.

Figure 18: Secure Virtual Machine Image Creation Process

Image Factories are automated processes and tools for assembling and customizing VM images. Consider them as the kitchen where the recipe (using the image sources) is followed to create the final VM image (the meal). Image factories ensure consistency and repeatability in the VM creation process,

with a strong focus on security. The concept is similar to continuous integration/continuous deployment (CI/CD) pipelines. Image factories could be a part of that, though this is not necessary.

Image factories serve as the assembly lines for VM images, where consistency, security, and efficiency are paramount. This can include:

- Building, testing, and fine-tuning VM images to ensure consistency across deployments
- Minimizing discrepancies that could lead to security vulnerabilities
- Streamlining the integration of security updates and configuration changes

Image Sources are the starting points for building a VM image. They provide the core components like the operating system, applications, libraries, and configuration files. Think of them as the ingredients for the VM recipe.

Image Sources focus on the careful curation and maintenance of the components that constitute VM images, including:

- Preserving a library of source code and settings essential for creating VM images
- Incorporating security checks within the build process
- Keeping a comprehensive version history for easy rollback in case of issues

Secure VM image creation encompasses a series of best practices aimed at reinforcing the security posture of VM images through the following.

- **Hardening**: To minimize potential vulnerabilities, set up VM images with only the essential software, services, and access rights.
- **Patch management**: Regularly update the VM images with the latest security improvements to protect against new threats.
- **Configuration management**: Use standardized templates and scripts to ensure all VM images meet the required security standards to automate the image creation workflow and reduce manual errors.
- **Validation and testing**: Before using a VM image, thoroughly check it for security weaknesses and operational issues to ensure it is safe and functions correctly. VM images must always come from trusted sources.

Ultimately, securing VM images is about creating a consistent and repeatable process that embeds security into the blueprint of VMs, ensuring that as new VMs are spun up, they are already equipped to resist cyber threats.

8.2.2.1 Recommended Tools & Best Practices for VMs

Leveraging the right tools is important to the success of any vulnerability management program. The following tools offer specialized functions that cater to the diverse needs of VM security.

- **Cloud Workload Protection Platforms (CWPP)** typically include features for in-depth vulnerability scans across cloud workloads (VMs, containers, serverless) and prioritize the findings based on exploitability and business impact.
- **Traditional vulnerability scanners** tend not to be as effective in the cloud, however many vulnerability scanners now also support agents. These products may be rebranded as CWPP, depending on the product.
- **Configuration management tools** automate patch deployment and configuration hardening.
- **Endpoint Detection and Response (EDR)** agents perform runtime monitoring and some support vulnerability assessment.
- **Security Information & Event Management (SIEM)** for real-time monitoring and reporting.

The following vulnerability management lifecycle represents a systematic approach to handling VM vulnerabilities, from discovery to resolution. In the cloud, this cycle should expand to cover images and alternatives to patching, like replacing running VMs with updated images (the concept behind immutability). The cycle is comprised of:

- **Identification**: Use automated tools to scan VMs for known vulnerabilities.
- **Assessment**: Analyze and evaluate the risk associated with identified vulnerabilities, considering the VM's role and the classification of data processed/stored, such as data sensitivity.
- **Mitigation and Reporting**: Apply patches, configure security settings, and employ workarounds to address vulnerabilities.
- **Documentation**: Keep detailed records of vulnerabilities, assessments, and remediation actions for reporting, compliance, and auditing.

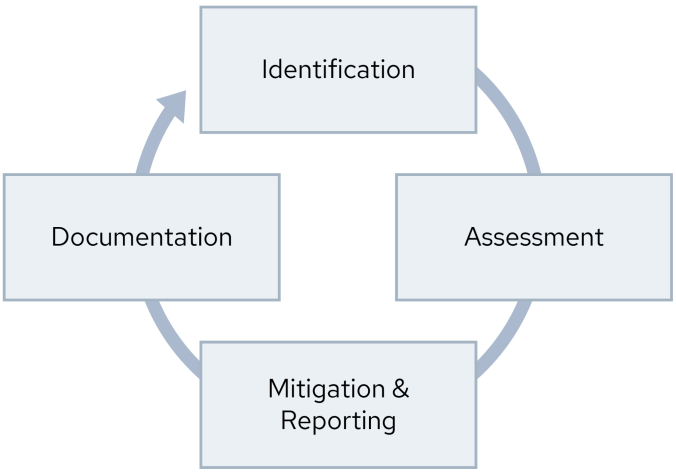


Figure 19: Vulnerability Management Lifecycle for VMs

By integrating these strategies, tools, and practices into the security framework, organizations can significantly enhance the protection of their VM environments against the many vulnerabilities that threaten digital assets in the contemporary cybersecurity landscape.

8.2.3 Snapshots & Public Exposures/Exfiltration

Snapshots are crucial for managing VM lifecycles, creating nearly instant copies of storage volumes for preservation and recovery. They capture the VM's state, including files and settings, and must be handled carefully to prevent unauthorized access and data leaks.

To mitigate risks:

- Implement stringent access controls, limiting snapshot creation and retrieval to trusted personnel.
- Encrypt snapshots to protect data even if exposed.
- Regularly review and delete unnecessary snapshots to enhance security and optimize storage.
- Use monitoring tools to detect unauthorized access.

Securing snapshots as rigorously as live systems prevents data breaches and maintains VM integrity.

8.3 Securing Containers

This section delves into the importance of building secure container images, orchestrating containers efficiently and safely, and managing the myriad security challenges that arise in containerized environments. It provides comprehensive insights into securing each step of the container lifecycle, from the creation of a container image to orchestrating its deployment with systems like Kubernetes.

More than with most other cloud infrastructure technologies, there is a wide variety of options to deploy containers in the cloud. This ranges from self-hosting Docker containers on a VM to consuming fully managed container orchestration services based on Kubernetes and everything in between.

As a result, the shared security responsibility allocation differs across these services. For many security-critical services, it must be clear whether they are self-managed, provider-managed, or orchestrator-managed. This includes identity management, host OS security (e.g., nodes in Kubernetes), software version updates, container image security, network orchestration, network policies, firewalls, load balancers, and so on.

8.3.1 Container Image Creation

A container image is a lightweight, executable package that includes everything needed to run an application, such as code, runtime, libraries, and settings. Created from instructions in a Dockerfile, it ensures consistency and portability across environments. Containers should use secure base images and immutable infrastructures, meaning once built and deployed, they are not modified but replaced with new images for updates. Secure artifact repositories are essential for storing these images safely.

8.3.2 Container Networking

Container networking is an extension of the host operating system (often Linux) networking.

Kubernetes networking, and therefore network isolation, happens on multiple levels ranging from individual containers to application-aware load balancers (e.g., Ingress Controller). Various technologies exist for defining network policies. Again, some of these may be offerings of a provider, while they could also be self-managed.

8.3.3 Container Orchestration & Management Systems

Container orchestration, essential for managing containerized applications, is exemplified by Kubernetes (often abbreviated to "K8s"), renowned for its flexibility and rich features. K8s automates application deployment, scaling, and management across machine clusters, ensuring consistent operation. Containers host microservices, maintaining uniform environments for application components. Leading CSPs like Amazon (EKS), Microsoft (AKS), and Google (GKE) have adopted Kubernetes, tailoring it to their platforms with proprietary enhancements. These customized versions offer users a mix of standard Kubernetes features and provider-specific improvements.

For managed Kubernetes, there are a variety of service models, with differing shared security responsibilities. For example, identities in Kubernetes cluster management can be Kubernetes native, or consumed from the provider.

The image on the following page shows a basic Kubernetes architecture with the core management components, two "pods" where the containers run, and a load balancer through which users access the deployed application.

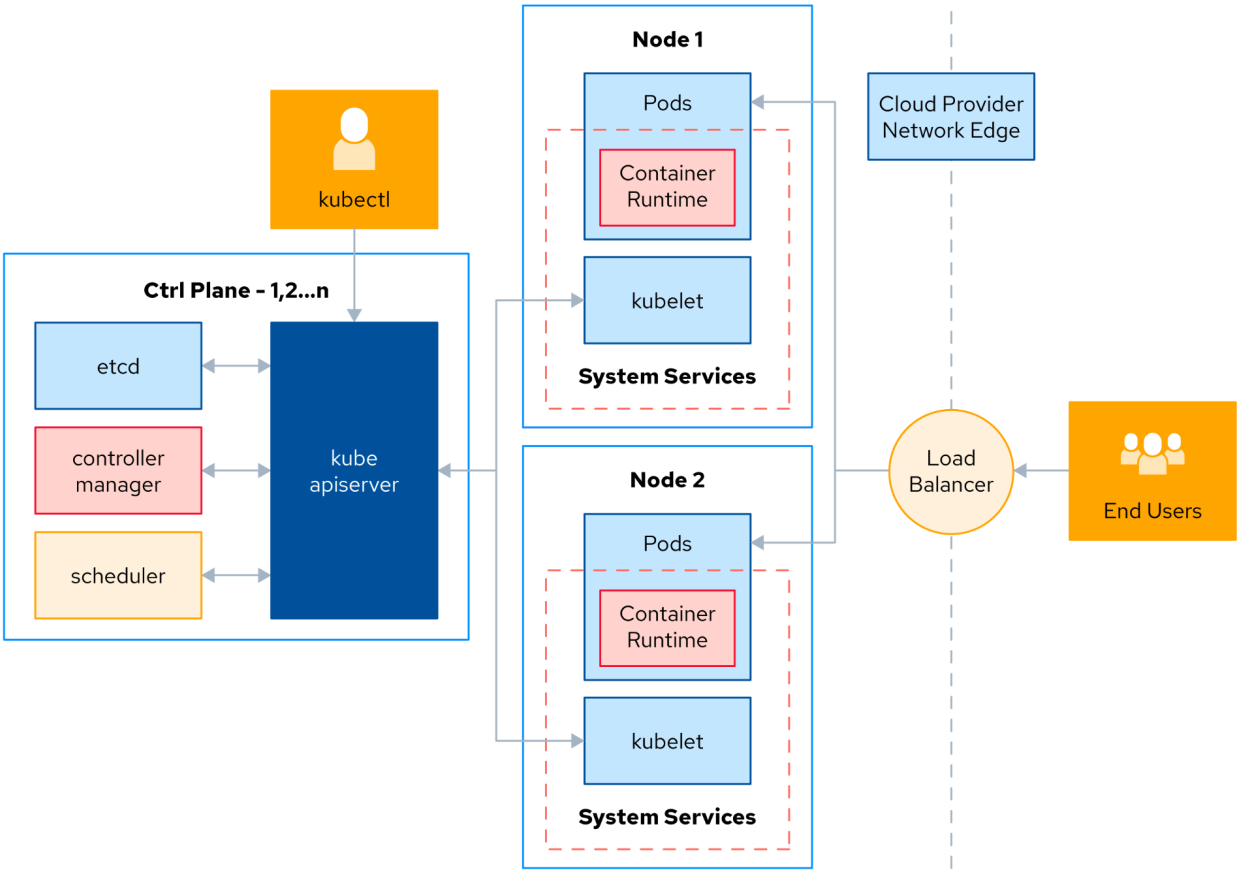


Figure 20: Basic Kubernetes Setup with Load Balancer and Pods

8.3.4 Container Orchestration Security

Securing Kubernetes and other container orchestration platforms involves:

- **Use CSP services:** Leverage cloud provider tools for security
- **Harden services:** Disable unnecessary features, use secure base images, and enforce network policies
- **Patch/update:** Regularly patch and update all components
- **Security policies:** Implement Kubernetes security and network policies
- **Benchmarks and tools:** Follow Center for Internet Security (CIS) benchmarks and use standardization tools
- **Secure image repository:** Use private repositories with rRBAC, image scanning, and signing
- **Configuration:** Start with robust, secure configurations

- **Host security:** Harden host operating systems
- **Storage security:** Encrypt data, use access controls, and monitor access
- **Network security:** Implement segmentation and firewall rules
- **Image validation:** Validate and sign images in the CI/CD pipeline[37]

8.3.4.1 Secure Artifact Repositories

Secure artifact repositories act as vaults for software components, including container images, ensuring that:

- Repositories enforce content trust mechanisms like digital signatures and verification to guarantee the authenticity and integrity of container images.
- Access is strictly controlled, allowing only verified users to push or pull images, much like a bank permitting transactions from only authenticated customers.
- Images are routinely scanned for vulnerabilities, akin to performing regular health check-ups to catch signs of illness early.
- Container images should be immutable.
- The provenance of images is thoroughly documented and protected, offering a clear lineage of their origins and makers, similar to a well-kept public registry.
- Secure artifact repositories seamlessly integrate with continuous integration and deployment pipelines to automate the scanning and validation of container images before deployment.

Best Practices for Using Secure Repositories

Securing artifact repositories involves several key practices:

- **Use secure sources:** Only use trusted images.
- **Sign and verify images:** Ensure authenticity with digital signatures.
- **Vulnerability scanning:** Scan images before deployment.
- **Access control:** Restrict repository access.
- **Audit trails:** Maintain records of access and modifications.
- **Regular updates:** Continuously update repository software.

[37] Additional material on CI/CD pipelines is provided in Domain 10: *Application Security*.

These measures ensure the integrity and security of container images, protecting applications from threats and ensuring compliance.

8.3.5 Runtime Protection for Containers

Runtime protection for containers ensures continuous security and smooth operation by detecting and managing threats in real-time. Key aspects include:

- **Real-time visibility:** Continuous monitoring for unusual behavior
- **Logging and auditing:** Detailed records for post-incident analysis
- **Micro-segmentation:** Isolating containers to contain breaches
- **Container-specific firewalls:** Managing network traffic flow
- **Automated responses:** Immediate actions to isolate threats and maintain system integrity

These proactive and reactive measures maintain the integrity and resilience of containerized applications.

8.4 Securing Serverless and Function as a Service

Serverless computing is a way for developers to write and deploy code without handling the underlying infrastructure. The cloud provider manages the servers, which includes provisioning them, scaling them to handle different loads, and maintaining them. This lets developers focus purely on coding, without worrying about the underlying work that goes into server management.

According to the Cloud Security Alliance (CSA) Serverless Working Group:

> *Serverless computing is a cloud-computing execution model in which the cloud provider is responsible for allocating compute and infrastructure resources needed to serve Application Owners' workloads. An Application Owner is no longer required to determine and control how many compute resources (and at what size) are allocated to serve their workload at any given time. It can rely on an abundance of computing resources that will be available to serve the workload on-demand. Therefore, serverless computing is offered under a* ***"Pay as you go"*** *paradigm where payment is generally made on actual physical resources like central processing unit (CPU) usage time.*

Note that various cloud providers also have products under this name that do not exactly match this definition. Examples include messaging and database services.

The term *serverless* is somewhat of a misnomer because servers are still used to run applications. However, managing these servers does not fall on the application owners. Instead, it is abstracted away by the CSP. This shift away from a traditional focus on infrastructure means developers only pay for the computing power they use, often billed down to the exact number of milliseconds of code execution, or a number of service invocations.

The primary advantage of serverless computing is its operational simplicity. Developers provide the code, and the cloud provider takes care of the rest, including all the operational aspects like system maintenance and scalability. The system automatically adjusts computing resources based on the application's needs, providing a highly flexible and efficient solution for developers who want to build and scale applications quickly and with minimal overhead.

In this section, we'll only discuss one popular subset of serverless: Functions as a Service (FaaS). Here, the cloud consumer defines a function and the events through which it will be triggered. The cloud provider handles the rest.

Each function in serverless is typically executed within a lightweight, single-use container that resides on a single-use virtual machine. This method ensures that each function invocation operates in a distinctly isolated environment, promoting strong isolation and preventing any interference between functions. The ephemeral nature of these execution environments significantly reduces the attack surface as there is no persistent operating system for the cloud consumer to manage, thereby minimizing potential security risks. Furthermore, this model enhances security by ensuring that each function's execution is both isolated and transient.

Nonetheless, developers must remember that, despite the cloud provider assuming many of the security responsibilities, they must still secure their application code, effectively manage access controls, and safeguard sensitive data that is transferred to and from the functions.

The figure illustrates the relationship between the function, container, and virtual machine in the FaaS model:

- The innermost circle represents the individual function, which contains the application code.
- The function is encapsulated within a container, which provides the necessary runtime environment and dependencies.
- The container is then executed on a virtual machine managed by the CSP.

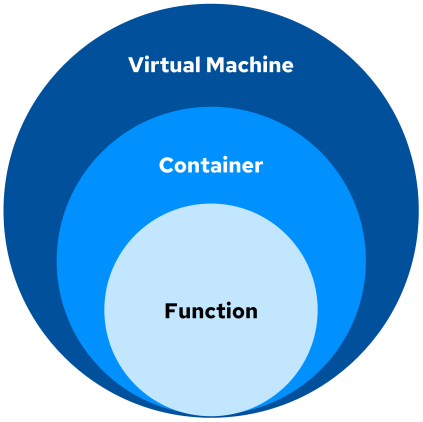


Figure 21: The FaaS Paradigm

8.4.1 FaaS Security Issues

In serverless computing, key security issues include:

- **Third-party services and APIs:** Increased attack surface if compromised
- **Vulnerable dependencies:** External libraries can harbor risks if not updated
- **Misconfigurations:** Overly permissive settings can expose sensitive resources
- **Overly-privileged IAM:** Excessive permissions heighten breach risks
- **Direct Internet access:** Functions exposed to external threats and data exfiltration

8.4.2 IAM for Serverless

IAM is crucial for securing serverless architectures. As these applications interact with various services, managing access becomes complex. Key IAM practices include:

- **Least privilege access**: Grant minimal necessary permissions and update regularly.
- **Fine-grained access control**: Specify precise permissions for individual functions or resources.
- **Context-aware authorization**: Use real-time attributes (e.g., user, device, time) to adapt access controls.
- **Immutable infrastructure and secret management**: Use secrets-management services, rotate credentials, and adopt immutable infrastructure principles.
- **Review and update IAM policies**: Regularly audit and adjust permissions to align with evolving needs.

8.4.3 Environment Variables & Secrets

Sensitive information handling in serverless applications requires caution. Use environment variables instead of hardcoding secrets like passwords or API keys into the code. These variables can be dynamically managed and injected at runtime, reducing exposure.

Cloud services like Amazon Web Services (AWS) Secrets Manager or Microsoft Azure Key Vault offer reliable mechanisms for secrets management, ensuring secure storage, retrieval, and rotation of credentials. Regularly rotating secrets minimizes the risk of compromised credentials. Controlling access through IAM roles ensures that only authorized entities can access or modify secrets.

Adopting these practices constructs a secure and efficient serverless environment, maintaining integrity and confidentiality. This aligns with a Zero Trust (ZT) security model, where trust is continually verified rather than assumed.

8.5 Securing AI Workloads

Artificial Intelligence (AI)[38] drives transformative changes in our lives and work by enabling machines to learn from data and simulate human intelligence. AI workloads include tasks from product recommendations to vehicle piloting, requiring extensive data, computational power, and specialized hardware like graphics processing units (GPUs) and tensor processing units (TPUs). These workloads benefit from cloud environments' dynamic scaling capabilities. AI helps industries automate tasks, enhance customer experiences, and solve complex problems. Understanding and managing AI workloads is crucial for maximizing AI's potential, with a focus on data, algorithms, and real-time processing to foster innovation.

[38] Additional information on AI is provided in Domain 12: *Related Technologies & Strategies*.

8.5.1 AI-System Threats

Here, we discuss risks and controls in the context of AI workloads.

CSA has developed the CSA Large Language Model (LLM) Threats Taxonomy, which establishes a common taxonomy and definitions for key terms related to risk scenarios and threats to Large Language Models (LLMs). The goal is to provide a shared language and conceptual framework to facilitate communication and alignment within the Industry. The document defines the following risk categories:

1. **Model manipulation**
2. **Data poisoning**
3. **Sensitive data disclosure**
4. **Model theft**
5. **Model Failure/malfunctioning**
6. **Insecure supply chain**
7. **Insecure apps/plugins**
8. **Denial of Service (DoS)**
9. **Loss of governance/compliance**

1. **Model Manipulation**
   This category involves attempts to evade detection or manipulate the LLM model to produce inaccurate or misleading results. It encompasses techniques, such as direct or indirect prompt injection (adversarial inputs), which aim to exploit vulnerabilities in the model's understanding and decision-making processes.

2. **Data Poisoning**
   Data poisoning refers to manipulating training data used to train an LLM model. This manipulation can be malicious, with attackers intentionally injecting false, misleading, or unintentional data points, where errors or biases in the original data set are included. In either case, data poisoning can lead to a tainted model that learns incorrect patterns, produces biased predictions, and becomes untrustworthy.

3. **Sensitive Data Disclosure**
   This category encompasses threats related to the unauthorized access, exposure, or leakage of sensitive information processed or stored by the LLM service. Sensitive data may include personal information, proprietary data, or confidential documents, the exposure of which could lead to privacy violations or security breaches.

4. **Model Theft**
   Model Theft (distillation) involves unauthorized access to, or replication of, the LLM model by malicious actors. Attackers may attempt to reverse-engineer the model architecture or extract proprietary algorithms and parameters, leading to intellectual property theft or the creation of unauthorized replicas.

5. **Model Failure/Malfunctioning**
   This category covers various types of failures or malfunctions within the LLM service, including software bugs, hardware failures, hallucinations, or operational errors. Such incidents can disrupt

service availability, degrade performance, or compromise the accuracy and reliability of the LLM model's outputs.

6. **Insecure Supply Chain**
   An insecure supply chain refers to vulnerabilities introduced through third-party components, dependencies, or services integrated into the LLM ecosystem. Vulnerabilities in the supply chain, such as compromised software libraries or hardware components, can be exploited to compromise the overall security and trustworthiness of the LLM service.

7. **Insecure Apps/Plugins**
   This category pertains to vulnerabilities introduced in plugins, functional calls, or extensions that interact with the LLM service. Insecure or maliciously designed applications/plugins may introduce security loopholes, elevate privilege levels, or facilitate unauthorized access to sensitive resources. Insecure plugins pose risks to both the input and output of integrated systems.

8. **Denial of Service (DoS)**
   DoS attacks aim to disrupt the availability or functionality of the LLM service by overwhelming it with a high volume of requests or malicious traffic and can render the service inaccessible to legitimate users, causing downtime, service degradation, or loss of trust.

9. **Loss of Governance/Compliance**
   This category involves the risk of non-compliance with regulatory requirements, industry standards, or internal governance policies governing the operation and use of the LLM service. Failure to adhere to governance and compliance standards can result in legal liabilities, financial penalties, or reputational damage.

These categories apply to LLMs but can be extended to generative AI (GenAI) in general. Addressing these risks requires a comprehensive approach that includes robust security measures integrated within the current security program,, ongoing risk assessments, threat intelligence integration, and proactive mitigation strategies tailored to an LLM/GenAI deployment's unique characteristics and challenges.

As we define strategies to secure AI workloads, it is important that all the assets within the AI-services supply chain, and all the steps of the AI-service lifecycle are taken into consideration.

In its Threat Taxonomy, CSA defines the following categories of Assets:

- Data Assets
- LLM Ops Environment
- Model
- Orchestrated Service
- AI Applications

And the following Lifecycle:

- Preparation
- Development
- Evaluation/Validation
- Deployment

- Delivery
- Service Retirement

8.5.2 AI Risk Mitigation and Shared Responsibilities

As the AI/machine learning (ML) landscape is developing very rapidly, any detailed discussion of AI security controls is part of the CSA AI Safety Initiative, but out of the scope of this guidance. However, there are some high-level principles in relation to where the responsibilities for these controls rests.

The AI ecosystem is complex, evolving, and multifaceted. It involves different actors/service/technology providers each of them having their own responsibility within the AI services supply chain. It is likely that we'll see the emergence of a shared responsibility model similar to the one introduced by cloud computing. Defining responsibility and allocating them between AI providers and consumers is crucial for a safe and effective AI strategy.

The following are some key AI system migration strategies grouped by category.

Data Security:

- *Encryption:* Protect data confidentiality during transmission and storage.
- *Differential privacy:* Introduce randomness into data or queries so that individual records can't be traced back to a person. It's like adding noise to a conversation to mask private details.
- *Secure multi-party computation:* Process data from multiple sources without exposing sensitive information by anonymizing or tokenizing sensitive information as part of the flows.
- *Confidential computing:* Use Trusted Execution Environments (TEEs) to safeguard data during processing and protect AI model execution.

Model Security:

- *Model hardening:* Defend against adversarial attacks to enhance model resilience.
- *Robust training:* Employ techniques to improve generalizability and reduce overfitting.[39]
- *Adversarial training:* Strengthen AI models against attacks by incorporating manipulated examples into their training data, enhancing their resilience, much like a fighter learning to counter different moves.
- *Model watermarking:* Embed unique identifiers to assert ownership and deter theft.
- *Output manipulation:* Altering the AI's responses to obscure its decision-making process can thwart potential thieves, much like a poker player's bluff.

[39] A scenario where a machine learning model learns the training data too well, including noise and outliers, resulting in poor generalization to new, unseen data, and potentially making the model vulnerable to adversarial attacks.

Infrastructure Security:

- *GPUs and TPUs:* To maintain system integrity, utilize hardware-based security features, regular firmware updates, and network security measures.
- *AI services:* Follow best practices for cloud services, including access controls and real-time monitoring.
- *Quotas and rate limiting:* Apply quotas and rate limiting to identify and prevent DoS and distributed denial-of-service (DDoS) attacks.

Supply Chain Security:

- *Policies:* Define and approve a cybersecurity policy for the supply chain.
- *Software supply chain risk management:* Regularly audit and update third-party dependencies.
- *Vetting third-party services:* Conduct security assessments before integration.
- *Trusted sources:* Rely on reputable sources for software dependencies, maintaining an approved list.

By proactively addressing these threats with the outlined strategies, organizations can fortify their AI infrastructure against current and emerging dangers, ensuring the resilience of their AI systems.[40]

[40] MITRE. (2024) *Adversarial Threat Landscape for Artificial-Intelligence Systems (ATLAS)* is a globally accessible, living knowledge base of adversary tactics and techniques against AI-enabled systems. It is based on real-world attack observations and realistic demonstrations from AI red teams and security groups and can help in the safeguarding of AI infrastructures.