AWS Security Blog

Hardening the RAG chatbot architecture powered by Amazon Bedrock: Blueprint for secure design and anti-pattern mitigation

by Magesh Dhanasekaran and Amy Tipple | on 06 AUG 2024 | in Advanced (300), Amazon Bedrock, Best Practices, Featured, Security, Identity, & Compliance, Technical How-to | Permalink | Comments | Share

This blog post demonstrates how to use Amazon Bedrock with a detailed security plan to deploy a safe and responsible chatbot application. In this post, we identify common security risks and anti-patterns that can arise when exposing a large language model (LLM) in an application. Amazon Bedrock is built with features you can use to mitigate vulnerabilities and incorporate secure design principles. This post highlights architectural considerations and best practice strategies to enhance the reliability of your LLM-based application.

Amazon Bedrock unleashes the fusion of generative artificial intelligence (AI) and LLMs, empowering you to craft impactful chatbot applications. As with technologies handling sensitive data and intellectual property, it's crucial that you prioritize security and adopt a robust security posture. Without proper measures, these applications can be susceptible to risks such as prompt injection, information disclosure, model exploitation, and regulatory violations. By proactively addressing these security considerations, you can responsibly use Amazon Bedrock foundation models and generative AI capabilities.

The chatbot application use case represents a common pattern in enterprise environments, where businesses want to use the power of generative AI foundation models (FMs) to build their own applications. This falls under the *Pre-trained models* category of the Generative AI Security Scoping Matrix. In this scope, businesses directly integrate with FMs like Anthropic's Claude through Amazon Bedrock APIs to create custom applications, such as customer support Retrieval Augmented Generation (RAG) chatbots, content generation tools, and decision support systems.

This post provides a comprehensive security blueprint for deploying chatbot applications that integrate with Amazon Bedrock, enabling the responsible adoption of LLMs and generative AI in enterprise environments. We outline mitigation strategies through secure design principles, architectural considerations, and best practices tailored to the challenges of integrating LLMs and generative AI capabilities.

By following the guidance in this post, you can proactively identify and mitigate risks associated with deploying and operating chatbot applications that integrate with Amazon Bedrock and use generative AI models. The guidance can help you strengthen the security posture, protect sensitive data and intellectual property, maintain regulatory compliance, and responsibly deploy generative AI capabilities within your enterprise environments.

This post contains the following high-level sections:

- Chatbot application architecture overview

Chatbot application architecture overview

The chatbot application architecture described in this post represents an example implementation that uses various AWS services and integrates with Amazon Bedrock and Anthropic's Claude 3 Sonnet LLM. This baseline architecture serves as a foundation to understand the core components and their interactions. However, it's important to note that there can be multiple ways for customers to design and implement a chatbot architecture that integrates with Amazon Bedrock, depending on their specific requirements and constraints. Regardless of the implementation approach, it's crucial to incorporate appropriate security controls and follow best practices for secure design and deployment of generative AI applications.

The chatbot application allows users to interact through a frontend interface and submit prompts or queries. These prompts are processed by integrating with Amazon Bedrock, which uses the Anthropic Claude 3 Sonnet LLM and a knowledge base built from ingested data. The LLM generates relevant responses based on the prompts and retrieved context from the knowledge base. While this baseline implementation outlines the core functionality, it requires incorporating security controls and following best practices to mitigate potential risks associated with deploying generative AI applications. In the subsequent sections, we discuss security anti-patterns that can arise in such applications, along with their corresponding mitigation strategies. Additionally, we present a secure and responsible architecture blueprint for the chatbot application powered by Amazon Bedrock.

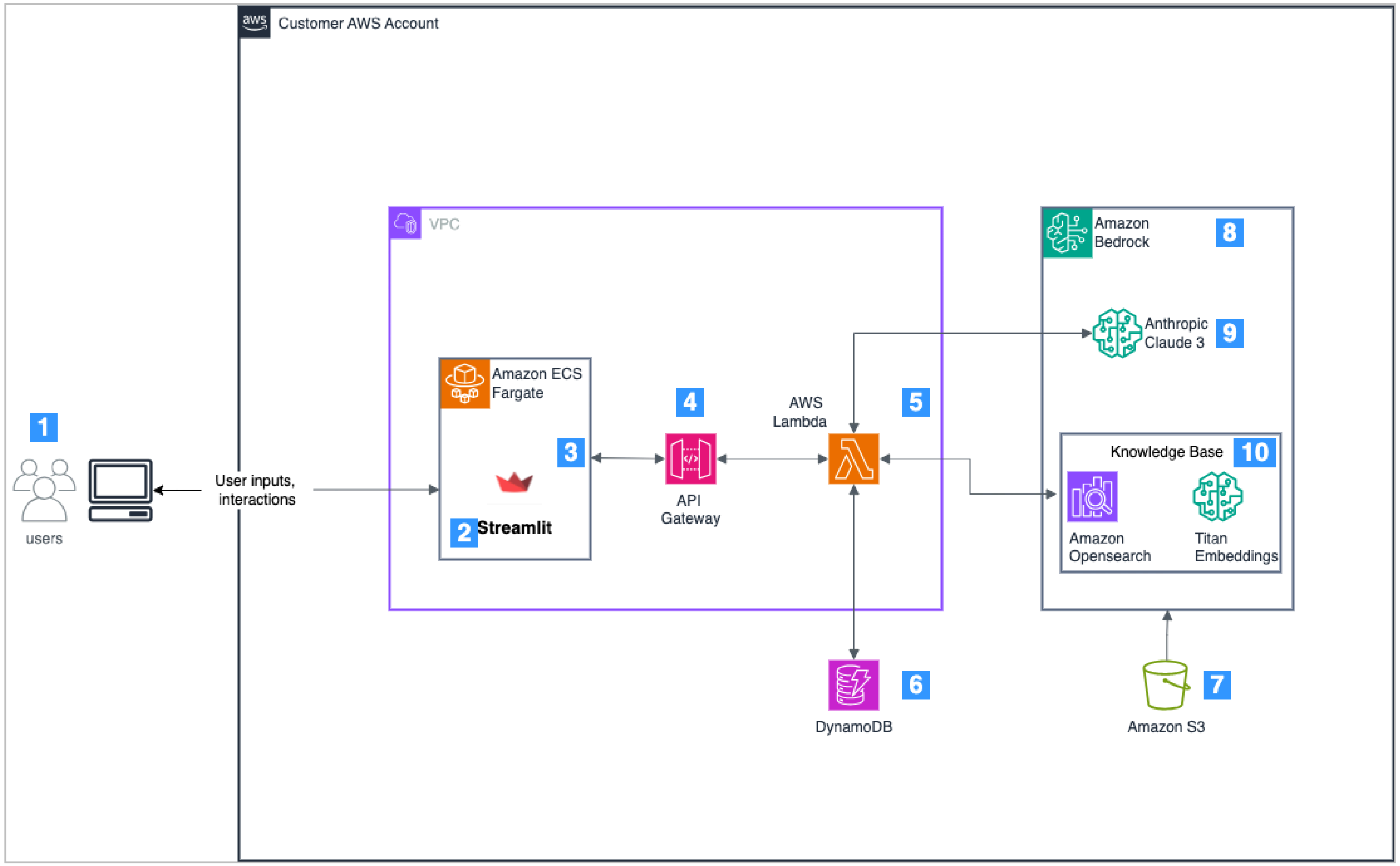


Figure 1: Baseline chatbot application architecture using AWS services and Amazon Bedrock

Components in the chatbot application baseline architecture

The chatbot application architecture uses various AWS services and integrates with the Amazon Bedrock service and Anthropic's Claude 3 Sonnet LLM to deliver an interactive and intelligent chatbot experience. The main components of the architecture (as shown in Figure 1) are:

1. **User interaction layer:**
   Users interact with the chatbot application through the Streamlit frontend (3), a Python-based open-source library, used to build the user-friendly and interactive interface.

2. Amazon Elastic Container Service (Amazon ECS) **on** AWS Fargate**:**
   A fully managed and scalable container orchestration service that eliminates the need to provision and manage servers, allowing you to run containerized applications without having to manage the underlying compute infrastructure.

3. **Application hosting and deployment:**
   The Streamlit application (3) components are hosted and deployed on Amazon ECS on AWS Fargate (2), maintaining scalability and high availability. This architecture represents the application and hosting environment in an independent virtual private cloud (VPC) to promote a loosely-coupled architecture. The Streamlit frontend can be replaced with your organization's specific frontend and quickly integrated with the backend Amazon API Gateway in the VPC. An application load balancer is used to distribute traffic to the Streamlit application instances.

4. **API Gateway driven Lambda Integration:**
   In this example architecture, instead of directly invoking the Amazon Bedrock service from the frontend, an API Gateway backed by an AWS Lambda function (5) is used as an intermediary layer. This approach promotes better separation of concerns, scalability, and secure access to Amazon Bedrock by limiting direct exposure from the frontend.

5. **Lambda**:
   Lambda provides highly scalable, short-term serverless compute. Here, the requests from Streamlit are processed. First, the history of the user's session is retrieved from Amazon DynamoDB (6). Second, the user's question, history, and the context are formatted into a prompt template and queried against Amazon Bedrock with the knowledge base, employing retrieval augmented generation (RAG).

6. **DynamoDB:**
   DynamoDB is responsible for storing and retrieving chat history, conversation history, recommendations, and other relevant data using the Lambda function.

7. **Amazon S3:**
   Amazon Simple Storage Services (Amazon S3) is a data storage service and is used here for storing data artifacts that are ingested into the knowledge base.

8. **Amazon Bedrock:**
   Amazon Bedrock plays a central role in the architecture. It handles the questions posed by the user using Anthropic Claude 3 Sonnet LLM (9) combined with a previously generated knowledge base (10) of the customer's organization-specific data.

9. **Anthropic Claude 3 Sonnet:**
   Anthropic Claude 3 Sonnet is the LLM used to generate tailored recommendations and responses based on user inputs and the context retrieved from the knowledge base. It's part of the text analysis and generation module in Amazon Bedrock.

10. **Knowledge base and data ingestion:**
    Relevant documents classified as public are ingested from Amazon S3 (9) into in an Amazon Bedrock knowledge base. Knowledge bases are backed by Amazon OpenSearch Service. Amazon Titan Embeddings (10) are used to generate the vector embeddings database of the documents. Storing the data as vector embeddings allows for semantic similarity searching of the documents to retrieve the context of the question posed by the user (RAG). By providing the LLM with context in addition to the question, there's a much higher chance of getting a useful answer from the LLM.

Comprehensive logging and monitoring strategy

This section outlines a comprehensive logging and monitoring strategy for the Amazon Bedrock-powered chatbot application, using various AWS services to enable centralized logging, auditing, and proactive monitoring of security events, performance metrics, and potential threats.

1. **Logging and auditing:**
    - AWS CloudTrail: Logs API calls made to Amazon Bedrock, including InvokeModel requests, as well as information about the user or service that made the request.
    - AWS CloudWatch Logs: Captures and analyzes Amazon Bedrock invocation logs, user prompts, generated responses, and errors or warnings encountered during the invocation process.
    - Amazon OpenSearch Service: Logs and indexes data related to the OpenSearch integration, context data retrievals, and knowledge base operations.
    - AWS Config: Monitors and audits the configuration of resources related to the chatbot application and Amazon Bedrock service, including IAM policies, VPC settings, encryption key management, and other resource configurations.
2. **Monitoring and alerting:**
    - AWS CloudWatch: Monitors metrics specific to Amazon Bedrock, such as the number of model invocations, latency of invocations, and error metrics (client-side errors, server-side errors, and throttling). Configures targeted CloudWatch alarms to proactively detect and respond to anomalies or issues related to Bedrock invocations and performance.
    - AWS GuardDuty: Continuously monitors CloudTrail logs for potential threats and unauthorized activity within the AWS environment.
    - AWS Security Hub: Provides centralized security posture management and compliance checks.
    - Amazon Security Lake: Provides a centralized data lake for log analysis; is integrated with CloudTrail and SecurityHub.
3. **Security information and event management integration:**

- Integrate with security information and event management (SIEM) solutions for centralized log management, real-time monitoring of security events, and correlation of logging data from multiple sources (CloudTrail, CloudWatch Logs, OpenSearch Service, and so on).

4. **Continuous improvement:**
   - Regularly review and update logging and monitoring configurations, alerting thresholds, and integration with security solutions to address emerging threats, changes in application requirements, or evolving best practices.

Security anti-patterns and mitigation strategies

This section identifies and explores common security anti-patterns associated with the Amazon Bedrock chatbot application architecture. By recognizing these anti-patterns early in the development and deployment phases, you can implement effective mitigation strategies and fortify your security posture.

Addressing security anti-patterns in the Amazon Bedrock chatbot application architecture is crucial for several reasons:

1. **Data protection and privacy:** The chatbot application processes and generates sensitive data, including personal information, intellectual property, and confidential business data. Failing to address security anti-patterns can lead to data breaches, unauthorized access, and potential regulatory violations.
2. **Model integrity and reliability:** Vulnerabilities in the chatbot application can enable bad actors to manipulate or exploit the underlying generative AI models, compromising the integrity and reliability of the generated outputs. This can have severe consequences, particularly in decision-support or critical applications.
3. **Responsible AI deployment:** As the adoption of generative AI models continues to grow, it's essential to maintain responsible and ethical deployment practices. Addressing security anti-patterns is crucial for maintaining trust, transparency, and accountability in the chatbot application powered by AI models.
4. **Compliance and regulatory requirements:** Many industries and regions have specific regulations and guidelines governing the use of AI technologies, data privacy, and information security. Addressing security anti-patterns is a critical step towards adhering to and maintaining compliance for the chatbot application.

The security anti-patterns that are covered in this post include:

1. Lack of secure authentication and access controls
2. Insufficient input validation and sanitization
3. Insecure communication channels
4. Inadequate prompt and response logging, auditing, and non-repudiation
5. Insecure data storage and access controls
6. Failure to secure FMs and generative AI components

7. Lack of responsible AI governance and ethics
8. Lack of comprehensive testing and validation

Anti-pattern 1: Lack of secure authentication and access controls

In a generative AI chatbot application using Amazon Bedrock, a lack of secure authentication and access controls poses significant risks to the confidentiality, integrity, and availability of the system. Identity spoofing and unauthorized access can enable threat actors to impersonate legitimate users or systems, gain unauthorized access to sensitive data processed by the chatbot application, and potentially compromise the integrity and confidentiality of the customer's data and intellectual property used by the application.

Identity spoofing and unauthorized access are important areas to address in this architecture, as the chatbot application handles user prompts and responses, which may contain sensitive information or intellectual property. If a threat actor can impersonate a legitimate user or system, they can potentially inject malicious prompts, retrieve confidential data from the knowledge base, or even manipulate the responses generated by the Anthropic Claude 3 LLM integrated with Amazon Bedrock.

Anti-pattern examples

- Exposing the Streamlit frontend interface or the API Gateway endpoint without proper authentication mechanisms, potentially allowing unauthenticated users to interact with the chatbot application and inject malicious prompts.
- Storing or hardcoding AWS access keys or API credentials in the application code or configuration files, increasing the risk of credential exposure and unauthorized access to AWS services like Amazon Bedrock or DynamoDB.
- Implementing weak or easily guessable passwords for administrative or service accounts with elevated privileges to access the Amazon Bedrock service or other critical components.
- Lacking multi-factor authentication (MFA) for AWS Identity and Access Management (IAM) users or roles with privileged access, increasing the risk of unauthorized access to AWS resources, including the Amazon Bedrock service, if credentials are compromised.

Mitigation strategies

To mitigate the risks associated with a lack of secure authentication and access controls, implement robust IAM controls, as well as continuous logging, monitoring, and threat detection mechanisms.

IAM controls:

- Use industry-standard protocols like OAuth 2.0 or OpenID Connect, and integrate with AWS IAM Identity Center or other identity providers for centralized authentication and authorization for the Streamlit frontend interface and AWS API Gateway endpoints.
- Implement fine-grained access controls using AWS IAM policies and resource-based policies to restrict access to only the necessary Amazon Bedrock resources, Lambda functions, and other components required for the chatbot application.
- Enforce the use of MFA for all IAM users, roles, and service accounts with access to critical components like Amazon Bedrock, DynamoDB, or the Streamlit application.

Continuous logging and monitoring and threat detection:

- See the Comprehensive logging and monitoring strategy section for guidance on implementing centralized logging and monitoring solutions to track and audit authentication events, access attempts, and potential unauthorized access or credential misuse across the chatbot application components and Amazon Bedrock service, as well as using CloudWatch, Lambda, and GuardDuty to detect and respond to anomalous behavior and potential threats.

Anti-pattern 2: Insufficient input sanitization and validation

Insufficient input validation and sanitization in a generative AI chatbot application can expose the system to various threats, including injection events, data tampering, adversarial events, and data poisoning events. These vulnerabilities can lead to unauthorized access, data manipulation, and compromised model outputs.

Injection events: If user prompts or inputs aren't properly sanitized and validated, a threat actor can potentially inject malicious code, such as SQL code, leading to unauthorized access or manipulation of the DynamoDB chat history data. Additionally, if the chatbot application or components process user input without proper validation, a threat actor can potentially inject and run arbitrary code on the backend systems, compromising the entire application.

Data tampering: A threat actor can potentially modify user prompts or payloads in transit between the chatbot interface and Amazon Bedrock service, leading to unintended model responses or actions. Lack of data integrity checks can allow a threat actor to tamper with the context data exchanged between Amazon Bedrock and OpenSearch, potentially leading to incorrect or malicious search results influencing the LLM responses.

Data poisoning events: If the training data or context data used by the LLM or chatbot application isn't properly validated and sanitized, bad actors can potentially introduce malicious or misleading data, leading to biased or compromised model outputs.

Anti-pattern examples

- Failure to validate and sanitize user prompts before sending them to Amazon Bedrock, potentially leading to injection events or unintended data exposure.

- Lack of input validation and sanitization for context data retrieved from OpenSearch, allowing malformed or malicious data to influence the LLM's responses.
- Insufficient sanitization of LLM-generated responses before displaying them to users, enabling potential code injection or rendering of harmful content.
- Inadequate sanitization of user input in the Streamlit application or Lambda functions, failing to remove or escape special characters, code snippets, or potentially malicious patterns, enabling code injection events.
- Insufficient validation and sanitization of training data or other data sources used by the LLM or chatbot application, allowing data poisoning events that can introduce malicious or misleading data, leading to biased or compromised model outputs.
- Allowing unrestricted character sets, input lengths, or special characters in user prompts or data inputs, enabling adversaries to craft inputs that bypass input validation and sanitization mechanisms, potentially causing undesirable or malicious outputs.
- Relying solely on deny lists for input validation, which can be quickly bypassed by adversaries, potentially leading to injection events, data tampering, or other exploit scenarios.

Mitigation strategies

To mitigate the risks associated with insufficient input validation and sanitization, implement robust input validation and sanitization mechanisms throughout the chatbot application and its components.

Input validation and sanitization:

- Implement strict input validation rules for user prompts at the chatbot interface and Amazon Bedrock service boundaries, defining allowed character sets, maximum input lengths, and disallowing special characters or code snippets. Use Amazon Bedrock's Guardrails feature, which allows defining denied topics and content filters to remove undesirable and harmful content from user interactions with your applications.
- Use allow lists instead of deny lists for input validation to maintain a more robust and comprehensive approach.
- Sanitize user input by removing or escaping special characters, code snippets, or potentially malicious patterns.

Data flow validation:

- Validate and sanitize data flows between components, including:
  - User prompts sent to the FM and responses generated by the FM and returned to the chatbot interface.
  - Training data, context data, and other data sources used by the FM or chatbot application.

Protective controls:

- Use AWS Web Application Firewall (WAF) for input validation and protection against common web exploits.
- Use AWS Shield for protection against distributed denial of service (DDoS) events.
- Use CloudTrail to monitor API calls to Amazon Bedrock, including InvokeModel requests.
- See the Comprehensive logging and monitoring strategy section for guidance on implementing Lambda functions, Amazon EventBridge rules, and CloudWatch Logs to analyze CloudTrail logs, ingest application logs, user prompts, and responses, and integrate with incident response and SIEM solutions for detecting, investigating, and mitigating security incidents related to input validation and sanitization, including jailbreaking attempts and anomalous behavior.

Anti-pattern 3: Insecure communication channels

Insecure communication channels between chatbot application components can expose sensitive data to interception, tampering, and unauthorized access risks. Unsecured channels enable man-in-the-middle events where threat actors intercept, modify data in transit such as user prompts, responses, and context data, leading to data tampering, malicious payload injection, and unauthorized information access.

Anti-pattern examples

- Failure to use AWS PrivateLink for secure service-to-service communication within the VPC, exposing communications between Amazon Bedrock and other AWS services to potential risks over the public internet, even when using HTTPS.
- Absence of data integrity checks or mechanisms to detect and prevent data tampering during transmission between components.
- Failure to regularly review and update communication channel configurations, protocols, and encryption mechanisms to address emerging threats and ensure compliance with security best practices.

Mitigation strategies

To mitigate the risks associated with insecure communication channels, implement secure communication mechanisms and enforce data integrity throughout the chatbot application's components and their interactions. Proper encryption, authentication, and integrity checks should be employed to protect sensitive data in transit and help prevent unauthorized access, data tampering, and man-in-the-middle events.

Secure communication channels:

- Use PrivateLink for secure service-to-service communication between Amazon Bedrock and other AWS services used in the chatbot application architecture. PrivateLink provides a private, isolated communication channel within the Amazon VPC, eliminating the need to traverse the public internet. This mitigates the risk of potential interception, tampering, or unauthorized access to sensitive data transmitted between services, even when using HTTPS.

- Use AWS Certificate Manager (ACM) to manage and automate the deployment of SSL/TLS certificates used for secure communication between the chatbot frontend interface (the Streamlit application) and the API Gateway endpoint. ACM simplifies the provisioning, renewal, and deployment of SSL/TLS certificates, making sure that communication channels between the user-facing components and the backend API are securely encrypted using industry-standard protocols and up-to-date certificates.

Continuous logging and monitoring:

- See the Comprehensive Logging and Monitoring Strategy section for guidance on implementing centralized logging and monitoring mechanisms to detect and respond to potential communication channel anomalies or security incidents, including monitoring communication channel metrics, API call patterns, request payloads, and response data, using AWS services like CloudWatch, CloudTrail, and AWS WAF.

Network segmentation and isolation controls

- Implement network segmentation by deploying the Amazon ECS cluster within a dedicated VPC and subnets, isolating it from other components and restricting communication based on the principle of least privilege.
- Create separate subnets within the VPC for the public-facing frontend tier and the backend application tier, further isolating the components.
- Use AWS security groups and network access control lists (NACLs) to control inbound and outbound traffic at the instance and subnet levels, respectively, for the ECS cluster and the frontend instances.

Anti-pattern 4: Inadequate logging, auditing, and non-repudiation

Inadequate logging, auditing, and non-repudiation mechanisms in a generative AI chatbot application can lead to several risks, including a lack of accountability, challenges in forensic analysis, and compliance concerns. Without proper logging and auditing, it’s challenging to track user activities, diagnose issues, perform forensic analysis in case of security incidents, and demonstrate compliance with regulations or internal policies.

Anti-pattern examples

- Lack of logging for data flows between components, such as user prompts sent to Amazon Bedrock, context data exchanged with OpenSearch, and responses from the LLM, hindering investigative efforts in case of security incidents or data breaches.
- Insufficient logging of user activities within the chatbot application—such as sign in attempts, session duration, and actions performed—limiting the ability to track and attribute actions to specific users.
- Absence of mechanisms to ensure the integrity and authenticity of logged data, allowing potential tampering or repudiation of logged events.

- Failure to securely store and protect log data from unauthorized access or modification, compromising the reliability and confidentiality of log information.

Mitigation strategies

To mitigate the risks associated with inadequate logging, auditing, and non-repudiation, implement comprehensive logging and auditing mechanisms to capture critical events, user activities, and data flows across the chatbot application components. Additionally, measures must be taken to maintain the integrity and authenticity of log data, help prevent tampering or repudiation, and securely store and protect log information from unauthorized access.

Comprehensive logging and auditing:

- See the Comprehensive logging and monitoring strategy section for detailed guidance on implementing logging mechanisms using CloudTrail, CloudWatch Logs, and OpenSearch Service, as well as using CloudTrail for logging and monitoring API calls, especially Amazon Bedrock API calls and other API activities within the AWS environment, using CloudWatch for monitoring Amazon Bedrock-specific metrics, and ensuring log data integrity and non-repudiation through the CloudTrail log file integrity validation feature and implementing S3 Object Lock and S3 Versioning for log data stored in Amazon S3.
- Make sure that log data is securely stored and protected from unauthorized access by using AWS Key Management Service (AWS KMS) for encryption at rest and implementing restrictive IAM policies and resource-based policies to control access to log data.
- Retain log data for an appropriate period based on compliance requirements, using CloudTrail log file integrity validation and CloudWatch Logs retention periods and data archiving capabilities.

User activity monitoring and tracking:

- Use CloudTrail for logging and monitoring API calls, especially Amazon Bedrock API calls and other API activities within the AWS environment, such as API Gateway, Lambda, and DynamoDB. Additionally, use CloudWatch for monitoring metrics specific to Amazon Bedrock, including the number of model invocations, latency, and error metrics (client-side errors, server-side errors, and throttling).
- Integrate with security information and event management (SIEM) solutions for centralized log management and real-time monitoring of security events.

Data integrity and non-repudiation:

- Implement digital signatures or non-repudiation mechanisms to verify the integrity and authenticity of logged data, minimizing tampering or repudiation of logged events. Use the CloudTrail log file integrity validation feature, which uses industry-standard algorithms (SHA-256 for hashing and SHA-256 with RSA for digital signing) to provide non-repudiation and verify log data integrity. For log data stored in Amazon S3, enable S3 Object Lock and S3 Versioning to provide an immutable, write once, read many (WORM) data storage model, helping to prevent object deletions or modifications, and maintaining data integrity and non-repudiation. Additionally, implement S3 bucket policies and IAM policies to restrict access to log data stored in S3, further enhancing the security and non-repudiation of logged events.

Anti-pattern 5: Insecure data storage and access controls

Insecure data storage and access controls in a generative AI chatbot application can lead to significant risks, including information disclosure, data tampering, and unauthorized access. Storing sensitive data, such as chat history, in an unencrypted or insecure manner can result in information disclosure if the data store is compromised or accessed by unauthorized entities. Additionally, a lack of proper access controls can allow unauthorized parties to access, modify, or delete data, leading to data tampering or unauthorized access.

Anti-pattern examples

- Storing chat history data in DynamoDB without encryption at rest using AWS KMS customer-managed keys (CMKs).
- Lack of encryption at rest using CMKs from AWS KMS for data in OpenSearch, Amazon S3, or other components that handle sensitive data.
- Overly permissive access controls or lack of fine-grained access control mechanisms for the DynamoDB chat history, OpenSearch, Amazon S3, or other data stores, increasing the risk of unauthorized access or data breaches.
- Storing sensitive data in clear text, or using insecure encryption algorithms or key management practices.
- Failure to regularly review and rotate encryption keys or update access control policies to address potential security vulnerabilities or changes in access requirements.

Mitigation strategies

To mitigate the risks associated with insecure data storage and access controls, implement robust encryption mechanisms, secure key management practices, and fine-grained access control policies. Encrypting sensitive data at rest and in transit, using customer-managed encryption keys from AWS KMS, and implementing least- privilege access controls based on IAM policies and resource-based policies can significantly enhance the security and protection of data within the chatbot application architecture.

Key management and encryption at rest:

- Implement AWS KMS to manage and control access to CMKs for data encryption across components like DynamoDB, OpenSearch, and Amazon S3.
  - Use CMKs to configure DynamoDB to automatically encrypt chat history data at rest.

  - Configure OpenSearch and Amazon S3 to use encryption at rest with AWS KMS CMKs for data stored in these services.
  - CMKs provide enhanced security and control, allowing you to create, rotate, disable, and revoke encryption keys, enabling better key isolation and separation of duties.
  - CMKs enable you to enforce key policies, audit key usage, and adhere to regulatory requirements or organizational policies that mandate customer-managed encryption keys.
  - CMKs offer portability and independence from specific services, allowing you to migrate or integrate data across multiple services while maintaining control over the encryption keys.
  - AWS KMS provides a centralized and secure key management solution, simplifying the management and auditing of encryption keys across various components and services.
- Implement secure key management practices, including:
  - Regular key rotation to maintain the security of your encrypted data.
  - Separation of duties to make sure that no single individual has complete control over key management operations.
  - Strict access controls for key management operations, using IAM policies and roles to enforce the principle of least privilege.

Fine-grained access controls:

- Implement fine-grained access controls for the DynamoDB chat history data store, OpenSearch, Amazon S3, and other data stores using IAM policies and roles.
- Implement fine-grained access controls and define least-privilege access policies for all resources handling sensitive data, such as the DynamoDB chat history data store, OpenSearch, Amazon S3, and other data stores or services. For example, use IAM policies and resource-based policies to restrict access to specific DynamoDB tables, OpenSearch domains, and S3 buckets, limiting access to only the necessary actions (for example, read, write, and list) based on the principle of least privilege. Extend this approach to all resources handling sensitive data within the chatbot application architecture, making sure that access is granted only to the minimum required resources and actions necessary for each component or user role.

Continuous improvement:

- Regularly review and update encryption configurations, access control policies, and key management practices to address potential security vulnerabilities or changes in access requirements.

Anti-pattern 6: Failure to secure FM and generative AI components

Inadequate security measures for FMs and generative AI components in a chatbot application can lead to severe risks, including model tampering, unintended information disclosure, and denial of service. Threat actors can manipulate unsecured FMs and generative AI models to generate biased, harmful, or malicious responses, potentially causing significant harm or reputational damage.

Lack of proper access controls or input validation can result in unintended information disclosure, where sensitive data is inadvertently included in model responses. Additionally, insecure FM or generative AI components can be vulnerable to denial-of-service events, disrupting the availability of the chatbot application and impacting its functionality.

Anti-pattern examples

- Insecure model fine tuning practices, such as using untrusted or compromised data sources, can lead to biased or malicious models.
- Lack of continuous monitoring for FM and generative AI components, leaving them vulnerable to emerging threats or known vulnerabilities.
- Lack of guardrails or safety measures to control and filter the outputs of FMs and generative AI components, potentially leading to the generation of harmful, biased, or undesirable content.
- Inadequate access controls or input validation for prompts and context data sent to the FM components, increasing the risk of injection events or unintended information disclosure.
- Failure to implement secure deployment practices for FM and generative AI components, including secure communication channels, encryption of model artifacts, and access controls.

Mitigation strategies

To mitigate the risks associated with inadequately secured foundational models (FMs) and generative AI components, implement secure integration mechanisms, robust model fine-tuning and deployment practices, continuous monitoring, and effective guardrails and safety measures. These mitigation strategies help prevent model tampering, unintended information disclosure, denial-of-service events, and the generation of harmful or undesirable content, while ensuring the security, reliability, and ethical alignment of the chatbot application’s generative AI capabilities.

<u>Secure integration with LLMs and knowledge bases:</u>

- Implement secure communication channels (for example HTTPS or PrivateLink) between Amazon Bedrock, OpenSearch, and the FM components to help prevent unauthorized access or data tampering.

- Implement strict input validation and sanitization for prompts and context data sent to the FM components to help prevent injection events or unintended information disclosure.
- Implement access controls and least-privilege principles for the OpenSearch integration to limit the data accessible to the LLM components.

Secure model fine tuning, deployment, and monitoring:

- Establish secure and auditable fine-tuning pipelines, using trusted and vetted data sources, to help prevent tampering or the introduction of biases.
- Implement secure deployment practices for FM and generative AI components, including access controls, secure communication channels, and encryption of model artifacts.
- Continuously monitor FM and generative AI components for security vulnerabilities, performance issues, and unintended behavior.
- Implement rate-limiting, throttling, and load-balancing mechanisms to help prevent denial-of-service events on FM and generative AI components.
- Regularly review and audit FM and generative AI components for compliance with security policies, industry best practices, and regulatory requirements.

Guardrails and safety measures

- Implement guardrails, which are safety measures designed to reduce harmful outputs and align the behavior of FMs and generative AI components with human values.
- Use keyword-based filtering, metric-based thresholds, human oversight, and customized guardrails tailored to the specific risks and cultural and ethical norms of each application domain.
- Monitor the effectiveness of guardrails through performance benchmarking and adversarial testing.

Jailbreak robustness testing

- Conduct jailbreak robustness testing by prompting the FMs and generative AI components with a diverse set of jailbreak attempts across different prohibited scenarios to identify weaknesses and improve model robustness.

Anti-pattern 7: Lack of responsible AI governance and ethics

While the previous anti-patterns focused on technical security aspects, it is equally important to address the ethical and responsible governance of generative AI systems. Without strong governance frameworks, ethical guidelines, and accountability measures, chatbot applications can result in unintended consequences, biased outcomes, and a lack of transparency and trust.

Anti-pattern examples

- Lack of an established ethical AI governance framework, including principles, policies, and processes to guide the responsible development and deployment of the generative AI chatbot application.
- Insufficient measures to ensure transparency, explainability, and interpretability of the LLM and generative AI components, making it difficult to understand and audit their decision-making processes.
- Absence of mechanisms for stakeholder engagement, public consultation, and consideration of societal impacts, potentially leading to a lack of trust and acceptance of the chatbot application.
- Failure to address potential biases, discrimination, or unfairness in the training data, models, or outputs of the generative AI system.
- Inadequate processes for testing, validation, and ongoing monitoring of the chatbot application's ethical behavior and alignment with organizational values and societal norms.

Mitigation strategies

To minimize a lack of responsible AI governance and ethics, establish a comprehensive ethical AI governance framework, promote transparency and interpretability, engage stakeholders and consider societal impacts, address potential biases and fairness issues, implement continuous improvement and monitoring processes, and use guardrails and safety measures. These mitigation strategies help to foster trust, accountability, and ethical alignment in the development and deployment of the generative AI chatbot application, mitigating the risks of unintended consequences, biased outcomes, and a lack of transparency.

<u>Ethical AI governance framework:</u>

- Establish an ethical AI governance framework, including principles, policies, and processes to guide the responsible development and deployment of the generative AI chatbot application.
- Define clear ethical guidelines and decision-making frameworks to address potential ethical dilemmas, biases, or unintended consequences.
- Implement accountability measures, such as designated ethics boards, ethics officers, or external advisory committees, to oversee the ethical development and deployment of the chatbot application.

<u>Transparency and interpretability:</u>

- Implement measures to promote transparency and interpretability of the LLM and generative AI components, allowing for auditing and understanding of their decision-making processes.

- Provide clear and accessible information to stakeholders and users about the chatbot application's capabilities, limitations, and potential biases or ethical considerations.

Stakeholder engagement and societal impact:

- Establish mechanisms for stakeholder engagement, public consultation, and consideration of societal impacts, fostering trust and acceptance of the chatbot application.
- Conduct impact assessments to identify and mitigate potential negative consequences or risks to individuals, communities, or society.

Bias and fairness:

- Address potential biases, discrimination, or unfairness in the training data, models, or outputs of the generative AI system through rigorous testing, bias mitigation techniques, and ongoing monitoring.
- Promote diverse and inclusive representation in the development, testing, and governance processes to reduce potential biases and blind spots.

Continuous improvement and monitoring:

- Implement processes for ongoing testing, validation, and monitoring of the chatbot application's behavior and alignment with organizational values and societal norms.
- Regularly review and update the AI governance framework, policies, and processes to address emerging ethical challenges, societal expectations, and regulatory developments.

Guardrails and safety measures:

- Implement guardrails, such as Guardrails for Amazon Bedrock, which are safety measures designed to reduce harmful outputs and align the behavior of LLMs and generative AI components with human values and responsible AI policies.
- Use Guardrails for Amazon Bedrock to define denied topics and content filters to remove undesirable and harmful content from interactions between users and your applications.
  - Define denied topics using natural language descriptions to specify topics or subject areas that are undesirable in the context of your application.
  - Configure content filters to set thresholds for filtering harmful content across categories such as hate, insults, sexuality, and violence based on your use cases and responsible AI policies.

- Use the personally identifiable information (PII) redaction feature to redact information such as names, email addresses, and phone numbers from LLM-generated responses or block user inputs that contain PII.

- Integrate Guardrails for Amazon Bedrock with CloudWatch to monitor and analyze user inputs and LLM responses that violate defined policies, enabling proactive detection and response to potential issues.
- Monitor the effectiveness of guardrails through performance benchmarking and adversarial testing, continuously refining and updating the guardrails based on real-world usage and emerging ethical considerations.

Jailbreak robustness testing:

- Conduct jailbreak robustness testing by prompting the LLMs and generative AI components with a diverse set of jailbreak attempts across different prohibited scenarios to identify weaknesses and improve model robustness.

Anti-pattern 8: Lack of comprehensive testing and validation

Inadequate testing and validation processes for the LLM system and the generative AI chatbot application can lead to unidentified vulnerabilities, performance bottlenecks, and availability issues. Without comprehensive testing and validation, organizations might fail to detect potential security risks, functionality gaps, or scalability and performance limitations before deploying the application in a production environment.

Anti-pattern examples

- Lack of functional testing to validate the correctness and completeness of the LLM's responses and the chatbot application's features and functionalities.
- Insufficient performance testing to identify bottlenecks, resource constraints, or scalability limitations under various load conditions.
- Absence of security testing, such as penetration testing, vulnerability scanning, and adversarial testing to uncover potential security vulnerabilities or model exploits.
- Failure to incorporate automated testing and validation processes into a continuous integration and continuous deployment (CI/CD) pipeline, leading to manual and one-time testing efforts that might overlook critical issues.
- Inadequate testing of the chatbot application's integration with external services and components, such as Amazon Bedrock, OpenSearch, and DynamoDB, potentially leading to compatibility issues or data integrity problems.

Mitigation strategies

To address the lack of comprehensive testing and validation, implement a robust testing strategy encompassing functional, performance, security, and integration testing. Integrate automated testing into a CI/CD pipeline, conduct security testing like threat modeling and penetration testing, and use adversarial validation techniques. Continuously improve testing processes to verify the reliability, security, and scalability of the generative AI chatbot application.

Comprehensive testing strategy:

- Establish a comprehensive testing strategy that includes functional testing, performance testing, load testing, security testing, and integration testing for the LLM system and the overall chatbot application.
- Define clear testing requirements, test cases, and acceptance criteria based on the application's functional and non-functional requirements, as well as security and compliance standards.

Automated testing and CI/CD integration:

- Incorporate automated testing and validation processes into a CI/CD pipeline, enabling continuous monitoring and assessment of the LLM's performance, security, and reliability throughout its lifecycle.
- Use automated testing tools and frameworks to streamline the testing process, improve test coverage, and facilitate regression testing.

Security testing and adversarial validation:

- Conduct threat modeling exercises early in the design process and as soon as the design is finalized for the chatbot application architecture to proactively identify potential security risks and vulnerabilities. Subsequently, conduct regular security testing—including penetration testing, vulnerability scanning, and adversarial testing—to uncover and validate identified security vulnerabilities or model exploits.
- Implement adversarial validation techniques, such as prompting the LLM with carefully crafted inputs designed to expose weaknesses or vulnerabilities, to improve the model's robustness and security.

Performance and load testing:

- Perform comprehensive performance and load testing to identify potential bottlenecks, resource constraints, or scalability limitations under various load conditions.
- Use tools and techniques for load generation, stress testing, and capacity planning to ensure the chatbot application can handle anticipated user traffic and workloads.

Integration testing:

- Conduct thorough integration testing to validate the chatbot application’s integration with external services and components, such as Amazon Bedrock, OpenSearch, and DynamoDB, maintaining seamless communication and data integrity.

Continuous improvement:

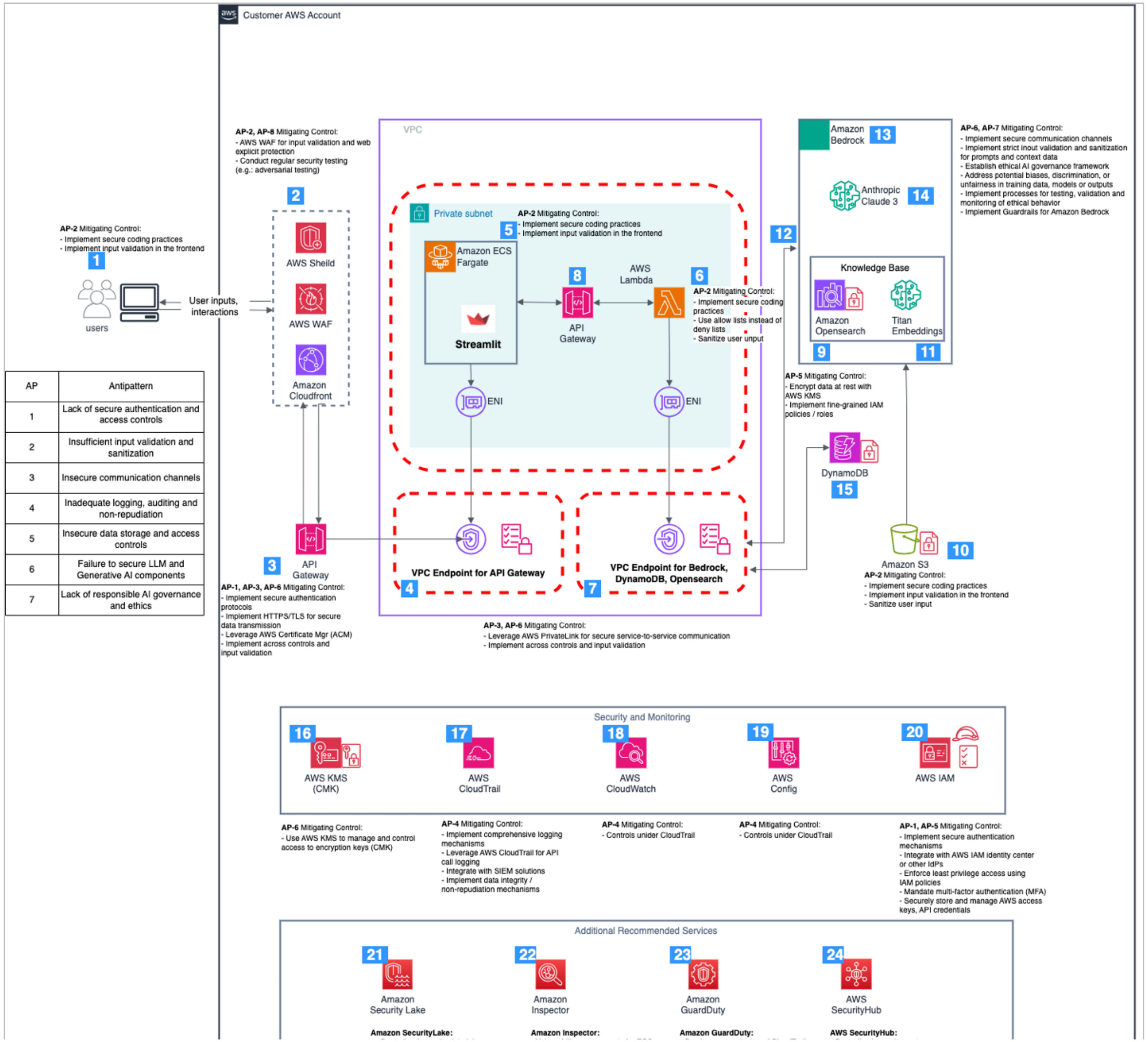
- Regularly review and update the testing and validation processes to address emerging threats, new vulnerabilities, or changes in application requirements.
- Use testing insights and results to continuously improve the LLM system, the chatbot application, and the overall security posture.

Common mitigation strategies for all anti-patterns

- Regularly review and update security measures, access controls, monitoring mechanisms, and guardrails for LLM and generative AI components to address emerging threats, vulnerabilities, and evolving responsible AI best practices.
- Conduct regular security assessments, penetration testing, and code reviews to identify and remediate vulnerabilities or misconfigurations related to logging, auditing, and non-repudiation mechanisms.
- Stay current with security best practices, guidance, and updates from AWS and industry organizations regarding logging, auditing, and non-repudiation for generative AI applications.

Secure and responsible architecture blueprint

After discussing the baseline chatbot application architecture and identifying critical security anti-patterns associated with generative AI applications built using Amazon Bedrock, we now present the secure and responsible architecture blueprint. This blueprint (Figure 2) incorporates the recommended mitigation strategies and security controls discussed throughout the anti-pattern analysis.



AP	Antipattern
1	Lack of secure authentication and access controls
2	Insufficient input validation and sanitization
3	Insecure communication channels
4	Inadequate logging, auditing and non-repudiation
5	Insecure data storage and access controls
6	Failure to secure LLM and Generative AI components
7	Lack of responsible AI governance and ethics

AP-2 Mitigating Control:
- Implement secure coding practices
- Implement input validation in the frontend

AP-2, AP-8 Mitigating Control:
- AWS WAF for input validation and web explicit protection
- Conduct regular security testing (e.g.: adversarial testing)

AP-1, AP-3, AP-6 Mitigating Control:
- Implement secure authentication protocols
- Implement HTTPS/TLS for secure data transmission
- Leverage AWS Certificate Mgr (ACM)
- Implement across controls and input validation

AP-2 Mitigating Control:
- Implement secure coding practices
- Implement input validation in the frontend

AP-2 Mitigating Control:
- Implement secure coding practices
- Use allow lists instead of deny lists
- Sanitize user unput

AP-3, AP-6 Mitigating Control:
- Leverage AWS PrivateLink for secure service-to-service communication
- Implement across controls and input validation

AP-6, AP-7 Mitigating Control:
- Implement secure communication channels
- Implement strict inout validation and sanitization for prompts and context data
- Establish ethical AI governance framework
- Address potential biases, discrimination, or unfairness in training data, models or outputs
- Implement processes for testing, validation and monitoring of ethical behavior
- Implement Guardrails for Amazon Bedrock

AP-5 Mitigating Control:
- Encrypt data at rest with AWS KMS
- Implement fine-grained IAM policies / roles

AP-2 Mitigating Control:
- Implement secure coding practices
- Implement input validation in the frontend
- Sanitize user input

AP-6 Mitigating Control:
- Use AWS KMS to manage and control access to encryption keys (CMK)

AP-4 Mitigating Control:
- Implement comprehensive logging mechanisms
- Leverage AWS CloudTrail for API call logging
- Integrate with SIEM solutions
- Implement data integrity / non-repudiation mechanisms

AP-4 Mitigating Control:
- Controls unider CloudTrail

AP-4 Mitigating Control:
- Controls unider CloudTrail

AP-1, AP-5 Mitigating Control:
- Implement secure authentication mechanisms
- Integrate with AWS IAM identity center or other IdPs
- Enforce least privilege access using IAM policies
- Mandate multi-factor authentication (MFA)
- Securely store and manage AWS access keys, API credentials

Amazon SecurityLake: **Amazon Inspector:** **Amazon GuardDuty:** **AWS SecurityHub:**

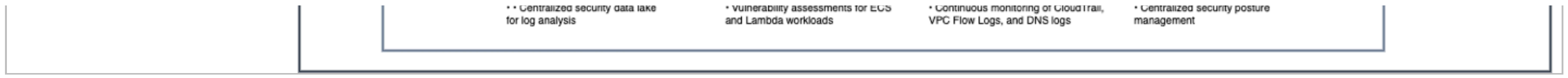

• Centralized security data lake for log analysis
• Vulnerability assessments for ECS and Lambda workloads
• Continuous monitoring of CloudTrail, VPC Flow Logs, and DNS logs
• Centralized security posture management

Figure 2: Secure and responsible generative AI chatbot architecture blueprint

In this target state architecture, unauthenticated users interact with the chatbot application through the frontend interface (1), where it’s crucial to mitigate the anti-pattern of insufficient input validation and sanitization by implementing secure coding practices and input validation. The user inputs are then processed through AWS Shield, AWS WAF, and CloudFront (2), which provide DDoS protection, web application firewall capabilities, and a content delivery network, respectively. These services help mitigate insufficient input validation, web exploits, and lack of comprehensive testing by using AWS WAF for input validation and conducting regular security testing.

The user requests are then routed through API Gateway (3), which acts as the entry point for the chatbot application, facilitating API connections to the Streamlit frontend. To address anti-patterns related to authentication, insecure communication, and LLM security, it’s essential to implement secure authentication protocols, HTTPS/TLS, access controls, and input validation within API Gateway. Communication between the VPC resources and API Gateway is secured through VPC endpoints (4), using PrivateLink for secure private communication and attaching endpoint policies to control which AWS principals can access the API Gateway service (8), mitigating the insecure communication channels anti-pattern.

The Streamlit application (5) is hosted on Amazon ECS in a private subnet within the VPC. It hosts the frontend interface and must implement secure coding practices and input validation to mitigate insufficient input validation and sanitization. User inputs are then processed by Lambda (6), a serverless compute service hosted within the VPC, which connects to Amazon Bedrock, OpenSearch, and DynamoDB through VPC endpoints (7). These VPC endpoints have endpoint policies attached to control access, enabling secure private communication between the Lambda function and the services, mitigating the insecure communication channels anti-pattern. Within Lambda, strict input validation rules, allow-lists, and user input sanitization are implemented to address the input validation anti-pattern.

User requests from the chatbot application are sent to Amazon Bedrock (12), a generative AI solution that powers the LLM capabilities. To mitigate the failure to secure FM and generative AI components anti-pattern, secure communication channels, input validation, and sanitization for prompts and context data must be implemented when interacting with Amazon Bedrock.

Amazon Bedrock interacts with OpenSearch Service (9) using Amazon Bedrock knowledge bases to retrieve relevant context data for the user’s question. The knowledge base is created by ingesting public documents from Amazon S3 (10). To mitigate the anti-pattern of insecure data storage and access controls, implement encryption at rest using AWS KMS and fine-grained IAM policies and roles for access control within OpenSearch Service. Titan Embeddings (11) are the format of the vector embeddings, which represent the documents stored in Amazon S3. The vector format enables similarity calculation and retrieval of relevant information (12). To address the failure to secure FM and generative AI components anti-pattern, secure integration with Titan Embeddings and input data validation should be implemented.

The knowledge base data, user prompts, and context data are processed by Amazon Bedrock (13) with the Claude 3 LLM (14). To address the anti-patterns of failure to secure FM and generative AI components, as well as lack of responsible AI governance and ethics, secure communication channels, input validation, ethical AI governance frameworks, transparency and interpretability measures, stakeholder engagement, bias mitigation, and guardrails like Guardrails for Amazon Bedrock should be implemented.

The generated responses and recommendations are then stored and retrieved in Amazon DynamoDB (15) by the Lambda function. To mitigate insecure data storage and access, encrypting data at rest with AWS KMS (16) and implement fine-grained access controls through IAM policies and roles.

Comprehensive logging, auditing, and monitoring mechanisms are provided by CloudTrail (17), CloudWatch (18), and AWS Config (19) to address the inadequate logging, auditing, and non-repudiation anti-pattern. See the Comprehensive logging and monitoring strategy section for detailed guidance on implementing comprehensive logging, auditing, and monitoring mechanisms using CloudTrail, CloudWatch, CloudWatch Logs, and AWS Config to address the inadequate logging, auditing, and non-repudiation anti-pattern; including logging API calls made to Amazon Bedrock service, monitoring Amazon Bedrock-specific metrics, capturing and analyzing Bedrock invocation logs, and monitoring and auditing the configuration of resources related to the chatbot application and Amazon Bedrock service.

IAM (20) plays a crucial role in the overall architecture and in mitigating anti-patterns related to authentication and insecure data storage and access. IAM roles and permissions are critical in enforcing secure authentication mechanisms, least privilege access, multi-factor authentication, and robust credential management across the various components of the chatbot application. Additionally, service control policies (SCPs) can be configured to restrict access to specific models or knowledge bases within Amazon Bedrock, preventing unauthorized access or use of sensitive intellectual property.

Finally, GuardDuty (21), Amazon Inspector (22), Security Hub (23), and Security Lake (24) have been included as additional recommended services to further enhance the security posture of the chatbot application. GuardDuty (21) provides threat detection across the control and data planes, Amazon Inspector (22) enables vulnerability assessments and continuous monitoring of Amazon ECS and Lambda workloads. Security Hub (23) offers centralized security posture management and compliance checks, while Security Lake (24) acts as a centralized data lake for log analysis, integrated with CloudTrail and SecurityHub.

Conclusion

By identifying critical anti-patterns and providing comprehensive mitigation strategies, you now have a solid foundation for a secure and responsible deployment of generative AI technologies in enterprise environments.

The secure and responsible architecture blueprint presented in this post serves as a comprehensive guide for organizations that want to use the power of generative AI while ensuring robust security, data protection, and ethical governance. By incorporating industry-leading security controls—such as secure authentication mechanisms, encrypted data storage, fine-grained access controls, secure communication channels, input validation and sanitization, comprehensive logging and auditing, secure FM integration and monitoring, and responsible AI guardrails—this blueprint addresses the unique challenges and vulnerabilities associated with generative AI applications.

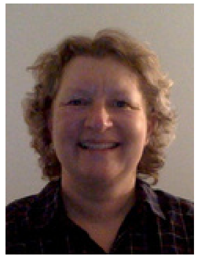
Moreover, the emphasis on comprehensive testing and validation processes, as well as the incorporation of ethical AI governance principles, makes sure that you can not only mitigate potential risks, but also promote transparency, explainability, and interpretability of the LLM components, while addressing potential biases and ensuring alignment with organizational values and societal norms.

By following the guidance outlined in this post and depicted in the architectural blueprint, you can proactively identify and mitigate potential risks, enhance the security posture of your generative AI-based chatbot solutions, protect sensitive data and intellectual property, maintain regulatory compliance, and responsibly deploy LLMs and generative AI technologies in your enterprise environments.

If you have feedback about this post, submit comments in the **Comments** section below. If you have questions about this post, contact AWS Support.

Magesh Dhanasekaran

Magesh is a Security Architect at AWS. He has a proven track record providing information security consulting services to financial institutions and government agencies in Australia and the United States. Magesh uses his experience in cloud security architecture, digital transformation, and secure application development practices to provide security advice on AWS products and services. He currently holds multiple industry certifications.



Amy Tipple

Amy is a Senior Data Scientist with the Professional Services Data and Machine Learning team and has been with AWS for approximately four years. Amy has worked on several engagements involving generative AI and is an advocate for making sure that generative AI-related security is accessible and understandable for AWS users.

TAGS: artificial intelligence, Security Blog

Like Share

Comments

Log in to comment