2025/01/18

JAWS-UGおおいた みんなで始めるBedrock Guardrails ハンズオン編

10分で紹介する Amazon Bedrock 利用時の セキュリティ対策

青柳 英明

自己紹介

氏名： 青柳 英明

所属： クラスメソッド 福岡オフィス

職種： AWS ソリューションアーキテクト

生成 AI エンジニア

JAWS-UG
AWS User Group - Japan
FUKUOKA

コアメンバーやってます

生成 AI 利用時のセキュリティ懸念

入力情報が学習に使われる？

情報漏洩・盗聴

・ユーザーが入力したテキスト
(質問内容に含まれている個人情報)

・生成 AI が出力したテキスト
(回答内容に含まれている機密情報)

・RAG で使用する社内ドキュメント

不正アクセス

・情報漏洩

・なりすまし

脆弱性を突いた攻撃

・サービス停止

・不正利用

生成 AI 利用時のセキュリティ懸念

入力情報が学習に使われる？

→ **Amazon Bedrock はユーザーのデータを学習に使いません！**

（ご安心を）

情報漏洩・盗聴

不正アクセス

脆弱性を突いた攻撃

→ 利用者が意識してセキュリティ対策を行う必要あり

（もちろん、セキュリティ対策は AWS のサービスを活用して実施できます）

生成 AI 利用時のセキュリティ対策

インフラ面の対策

・暗号化 (データ保管時、データ転送時)

・アクセス管理

・ネットワーク

・その他

生成 AI 観点の対策

・代表的なセキュリティリスク「OWASP Top 10 for LLM Applications」

　・プロンプトインジェクションへの対策

　　など

暗号化： データ保管時の暗号化

LLM： ステートレスな API

→ データは保存されないため、データ暗号化の検討不要

生成 AI アプリケーションにおいて保存されるデータ

・会話履歴

・RAG (ベクトルデータベース、データソース)

→ 暗号化の検討の対象

・AWS の各サービスで標準提供されている暗号化の仕組みを使う

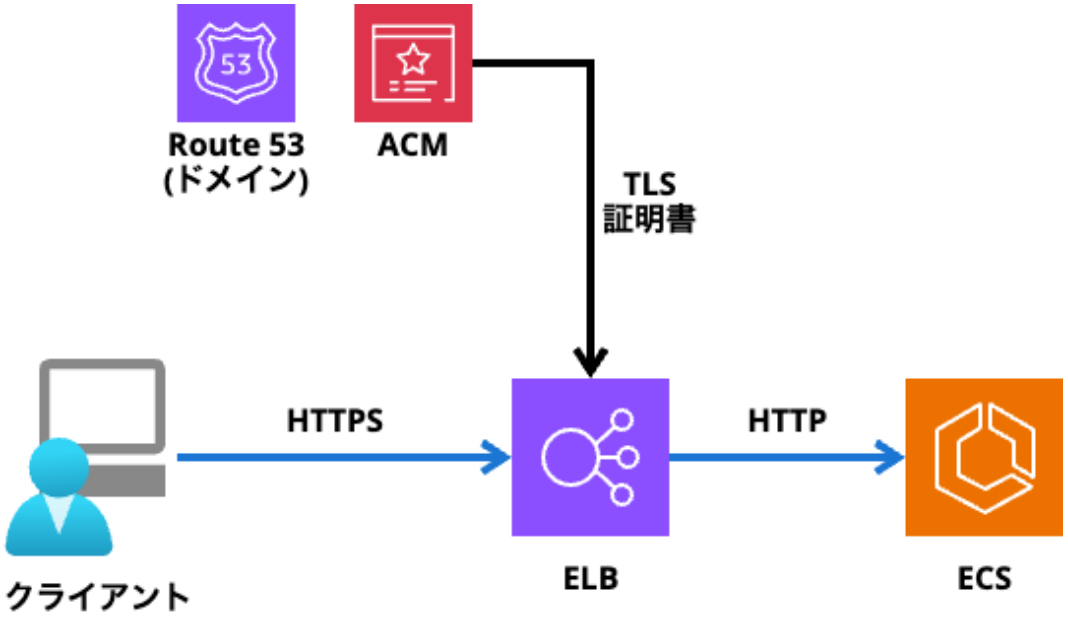
(S3、DynamoDB、etc.)

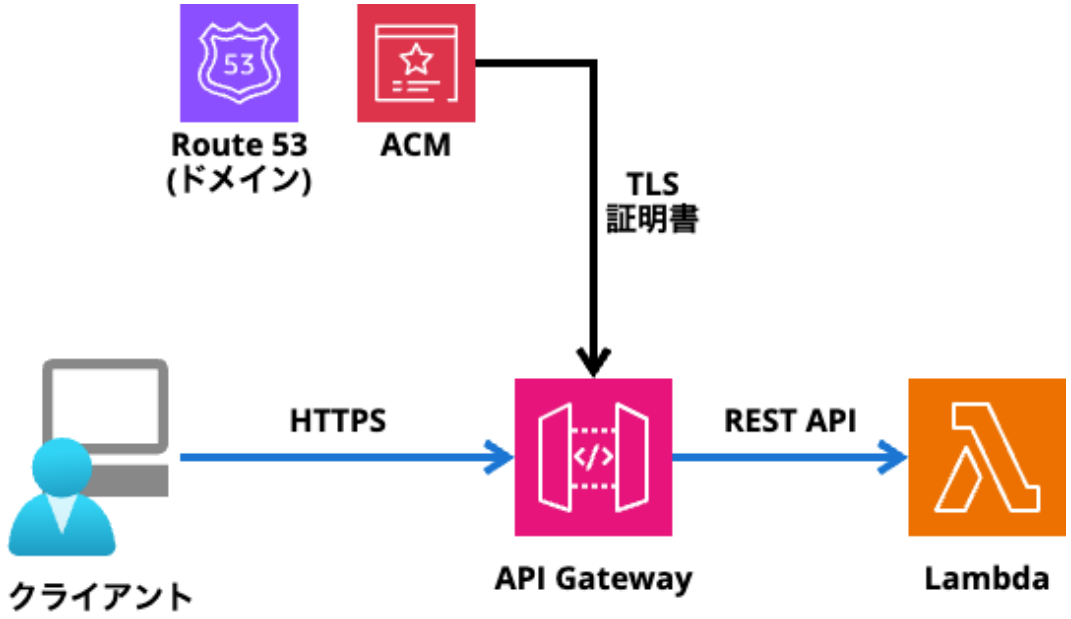
暗号化： データ転送時の暗号化

クライアント ～ アプリ間の通信： HTTPS

・ACM (AWS Certificate Manager) を使って無料で証明書を発行できる

・Route 53 を組み合わせることで独自ドメインでの HTTPS 運用も可能





暗号化： データ転送時の暗号化

アプリ ～ サービス間の通信： TLS

・デフォルトで TLS が使用される

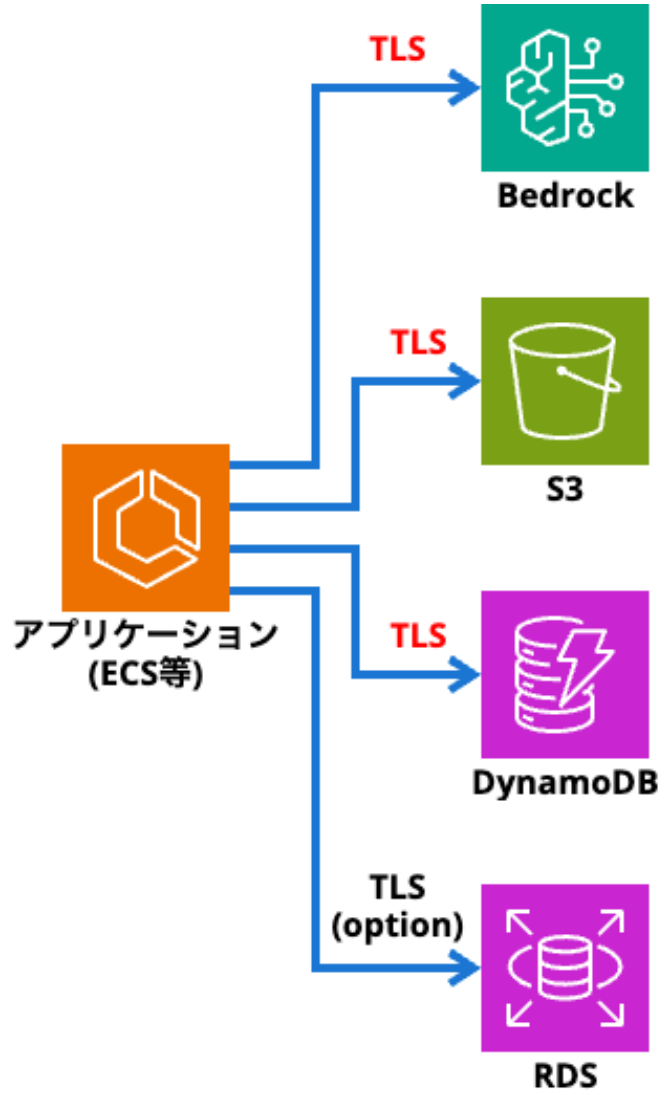
　・Bedrock、S3、DynamoDB 等

・設定により TLS を使用可能

　・RDS、EFS 等

・使用される TLS のバージョンに気をつける

　・現在の推奨は TLS 1.2



アクセス管理： クライアント → アプリ

アプリケーションへアクセスするユーザーの認証を行い、

許可されたユーザーのみに利用を許可する

社内ユーザーの認証： SAML ベース

・Active Directory (ADFS)、Entra ID、Google Workspace、etc.

社外ユーザーの認証： OpenID Connect ベース

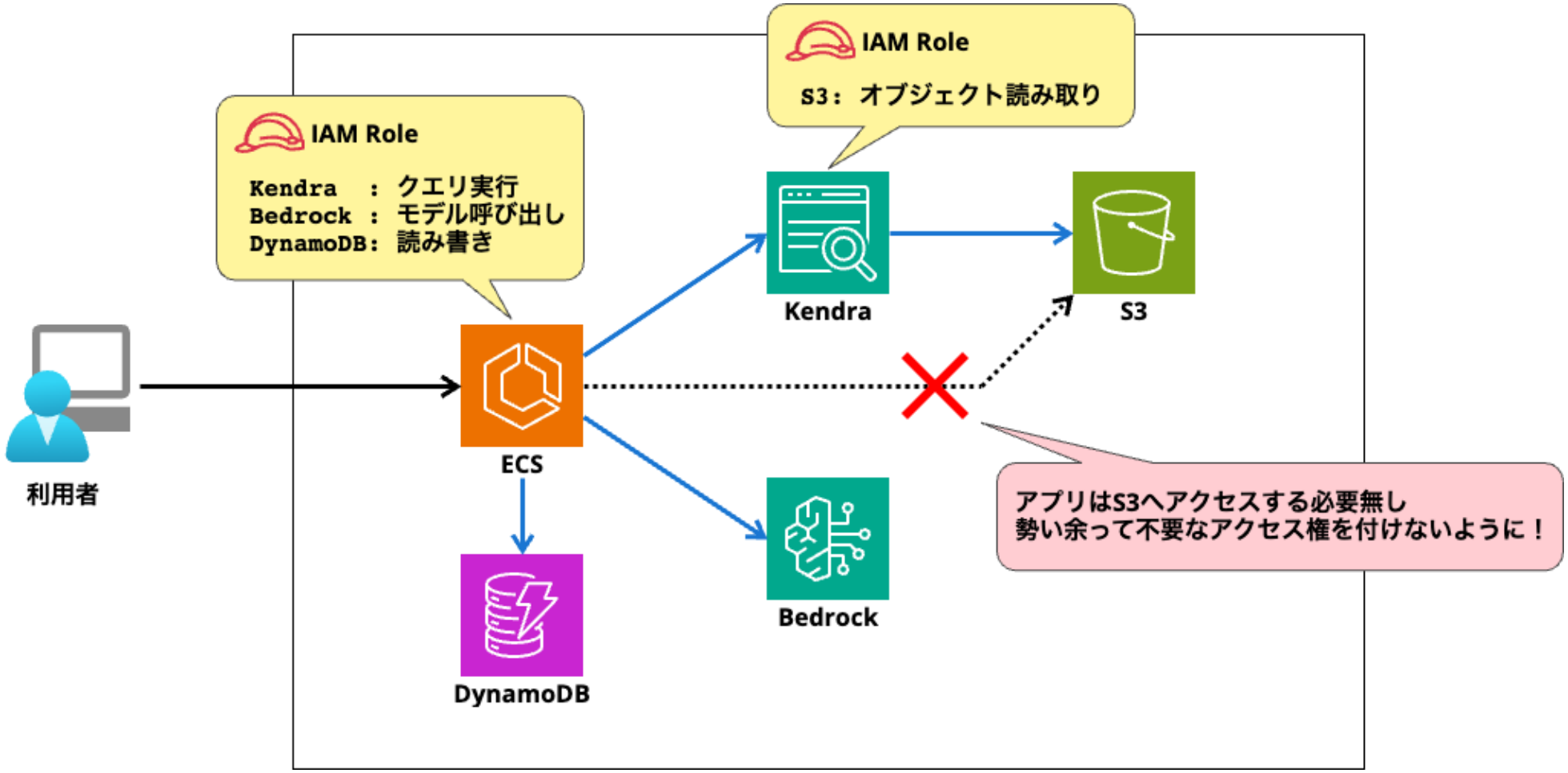
・Apple、Facebook、X、GitHub、etc.

アプリケーションへのユーザー認証の実装

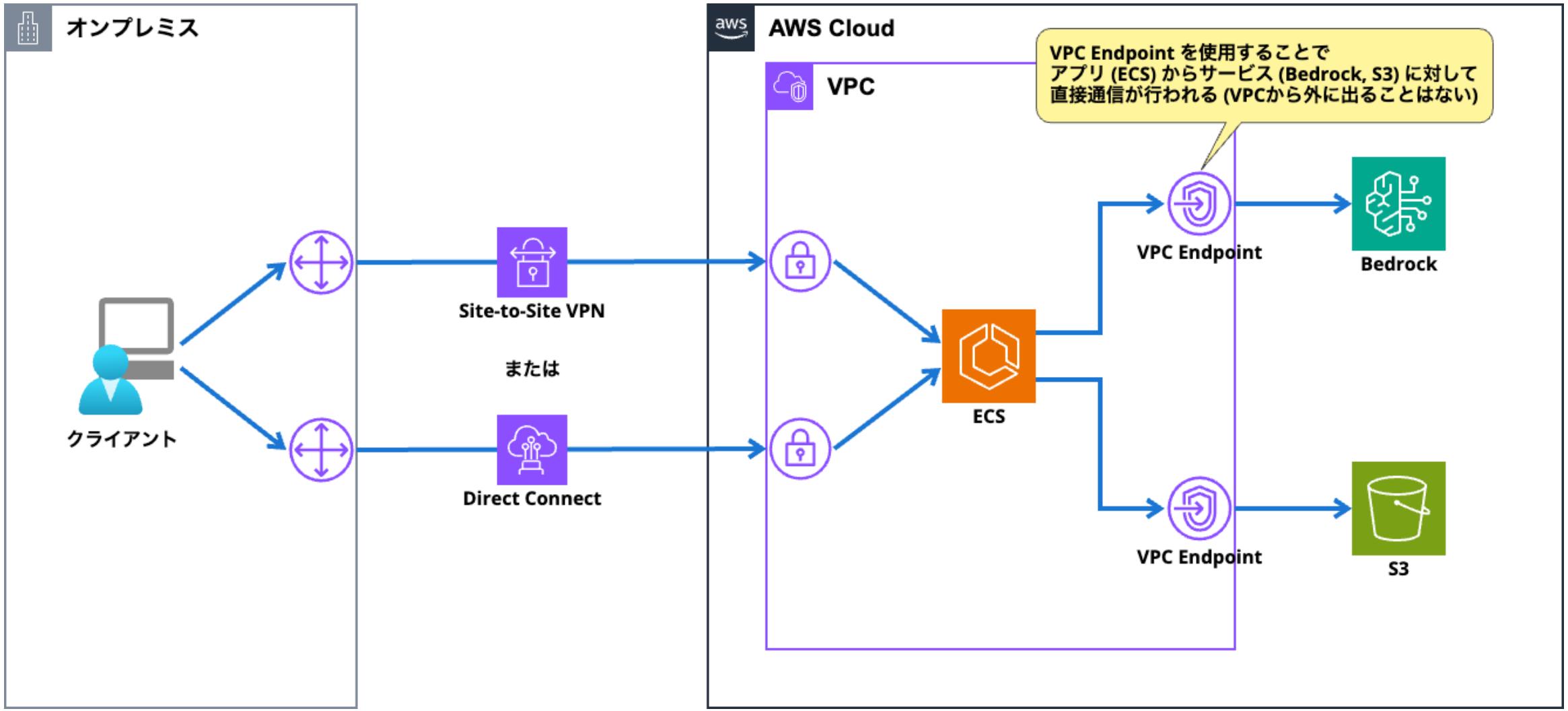
IAM Identity Center、Cognito、サードパーティ製ツールなどを使用

アクセス管理： アプリ内 (AWSサービス間)

「IAM ロール」を使ってアクセス管理



ネットワーク： 閉域化によるセキュリティ向上



ネットワーク： 閉域化は必須か？

標準的なネットワーク構成でも、セキュリティは担保されます！

暗号化： クライアント～アプリ間

・インターネットを経由するが HTTPS で暗号化されている

暗号化： アプリ内 (AWSサービス間)

・AWS 内に閉じた通信であり、かつ、TLS による暗号化も行われる

アクセス管理

閉域化を検討すべき場面：

・企業／システムのセキュリティポリシーで定められている

・コスト (利用費、運用コスト) が増加しても強固なセキュリティが必要

その他のインフラ面のセキュリティ対策

ロギング

・AWS サービスやアプリのロギングを有効にする

　　→ インシデント発生時に追跡が行えるようにしておく

シークレット（機微情報）の取り扱い

・OpenAI の API Key など機微情報の取り扱いを厳密にする

　　→ アプリに直書きしない、Secrets Manager などを利用する

GuardDuty、Security Hub などのセキュリティ対策サービスの活用

・人力による監視や対処は漏れや対応の遅れを生じさせる

生成 AI 観点のセキュリティリスク・対策

OWASP Top 10 for LLM Applications 2025
https://genai.owasp.org/resource/owasp-top-10-for-llm-applications-2025/

- 01: Prompt Injection (プロンプトインジェクション)
- 02: Sensitive Information Disclosure (機微情報の漏えい)
- 03: Supply Chain (サプライチェーン)
- 04: Data and Model Poisoning (データとモデルの汚染)
- 05: Inproper Output Handling (不適切な出力ハンドリング)
- 06: Excessive Agency (過剰な代理行為)
- 07: System Prompt Leakage (システムプロンプトの漏えい)
- 08: Vector and Embedding Weaknesses (ベクトルと埋め込みの弱点)
- 09: Misinformation (誤情報)
- 10: Unbounded Consumption (無制限の消費)

プロンプトインジェクション

悪意を持った者が LLM を意図的に誤動作させる指示を与え、

好ましくない内容を生成 AI に出力させる攻撃

(例) ・犯罪行為に使うことができる情報を回答させる (ex. 爆弾の作り方)

・個人情報、機密情報を聞き出す

LLM 側で対策は行なっているものの、特定の指示の与え方をすることで

対処策をすり抜けて攻撃が行われることがある

プロンプトインジェクション

プロンプトインジェクションを用いた攻撃手法：

・攻撃者が LLM に対して直接プロンプトインジェクションを試みる

・プロンプトインジェクションを含む情報 (ファイル、Webページ等) を
ユーザーの目につく場所に置き、ユーザーが LLM へ指示を行うように
仕向ける (間接的な攻撃)

(例) 一見普通の内容に見える Web ページだが、ユーザーがコピペして
「要約して」と指示するとプロンプトインジェクションが発動する

プロンプトインジェクション

対策

・システムプロンプトに「個人情報を出力しない」「有害な情報を出力しない」等の指示を与える

→ 攻撃者はプロンプトインジェクションによってシステムプロンプトを無視するように仕向けるため、効果が無い場合もある

・LLM への入力、LLM からの出力をチェックする

→ サードパーティ製ツール、 **Amazon Bedrock Guardrails**

・最新バージョン／最新リリースの LLM を利用する

不適切な出力ハンドリング

LLM の出力を精査せずに利用したため引き起こされる問題

・LLM が出力したサンプルコードをシェルで実行した結果、
ファイルの誤削除など重篤な問題が発生してしまう

・LLM が出力した HTML や JavaScript をブラウザが解釈した結果、
クロスサイトスクリプティング (XSS) が引き起こされてしまう

不適切な出力ハンドリング

対策

・LLM が出力したコードは、必ず人の手でチェック／レビューを行う

・LLM の出力に対して検証やサニタイズを行う

・LLM の出力をそのまま表示せずエンコード／エスケープする

→ 検証やエンコードを行うようにプロンプトで指示を与えることもできるが、
LLM は完全ではないため抜け漏れが発生することもある
後処理で行うのが確実