

トピック14E

デプロイと自動化のコンセプトを要約する



対象試験範囲

2.3セキュアなアプリケーションの開発、デプロイ、自動化に関するコンセプトを要約することができます。

ほとんどの組織は、継続的インテグレーション、デリバリー、およびデプロイの開発プロセスを含むアジャイル手法を使用しています。セキュアな開発・ステージング環境の作成と使用に加え、プロビジョニングツールとデプロビジョニングツールの使用をサポートできるようになる必要があります。

アプリケーションの開発、デプロイ、および自動化

DevSecOps文化はプロジェクトチームに、開発、セキュリティ、および運用の専門知識と経験に関する幅広い基盤を提供します。これにより、セキュリティタスクが自動化をより多く利用する環境が生まれます。自動化は、人間が介入することなく管理タスクを完了することです。タスク自動化手順は、GUIコントロールパネル、コマンドライン、またはスクリプトが呼び出すAPIを通じて構成できます。タスクを自動化することで、リソースの供給、アカウントの追加、権限の割り当て、インシデントの検知と対応、およびその他のネットワークセキュリティタスクをいくつでも実行できます。

手動構成はエラーが発生する可能性が高くなります。技術者がベストプラクティスを熟知していない、あるいは文書化が行われていない可能性があります。すると時間が経つにつれ、インスタンスとサービスの構成方法に小さな食い違いが多数生じます。ITおよびクラウドインフラストラクチャの保守、更新、保護においては、そうした小さな食い違いが大問題になることもあります。自動化はスケーラビリティ（拡張性）とエラスティシティ（弾力性）を改善します。

- スケーラビリティ（拡張性）** は、より多くのユーザーにサービスを提供するコストが、直線的であることを意味します。例えば、拡張可能なシステムのユーザー数が2倍になったとすれば、同レベルのサービスを維持するのにかかるコストも2倍（または2倍未満）になるということです。コストが2倍を超えると、システムの拡張性は低いことになります。
- エラスティシティ（弾力性）** は、リアルタイムかつオンデマンドで変更を処理するシステムの能力を指します。高いエラスティシティを有するシステムでは、需要が突然2倍（または3倍ないし4倍）になったとしても、サービスやパフォーマンスが低下することはありません。逆に、需要が少ないとときにコストを下げられることが重要な場合もあります。エラスティシティは、クラウドサービスの一般的なセールスポイントです。24時間年中無休でクラウドリソースを実行するのではなく、そのリソースへの需要が少ない場合、処理能力を減らしたり完全にシャットダウンができるのです。需要が再び増加すれば、求められる水準までリソースを増やせます。これによって費用効果の高い運用が可能になります。

セキュアなアプリケーション開発環境

アプリケーションまたは自動化の設計プロセスでは、セキュリティが重要な構成要素でなければなりません。フォームとスクリプトの単純な組み合わせでさえも、そのスクリプトがきちんと記述されていなければWebサーバーの脆弱性を生み出す恐れがあります。**ソフトウェア開発ライフサイクル(SDLC)**は、ソフトウェアの制作と保守を個別のフェーズに分割します。主要なSDLCとして、**ウォーターフォールモデル**と**アジャイル開発**の2つがあります。いずれのモデルも、開発プロジェクトの成功に対する、要求事項の分析と品質保証プロセスの重要性を強調するものです。

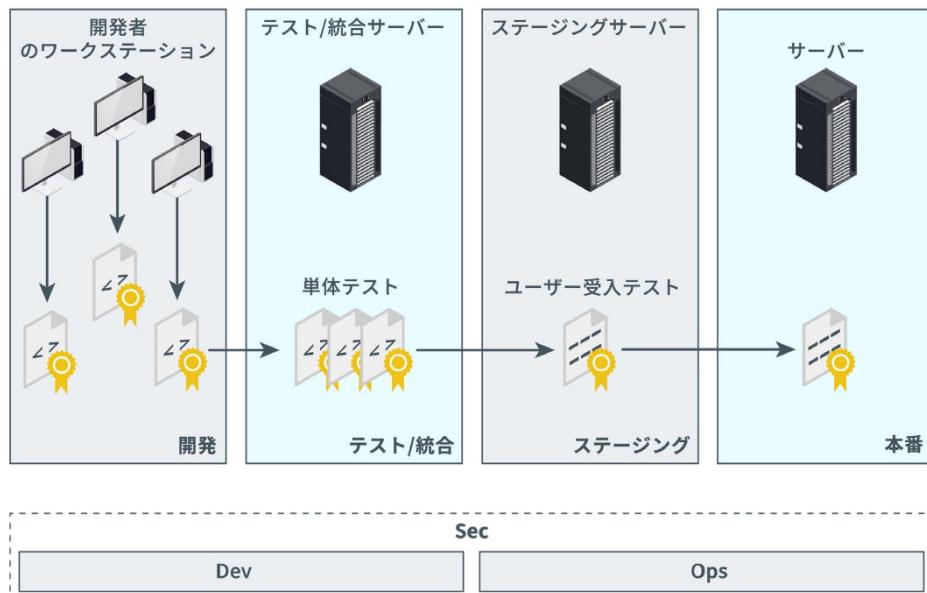
品質保証(QA)

品質保証プロセスとは、あるシステムが一連の要求事項や期待される事項に準拠しているかどうかを確認すべく、組織がそのシステムをテストすることです。これらの要求事項と期待値は、リストベースの評価、または業界規制や企業の定める品質基準など、内部および外部のコンプライアンス要素によって推進されます。品質管理(QC)は、あるシステムに欠陥や欠落がないかどうかを確かめるプロセスです。QC手順自体が**品質保証(QA)**プロセスによって定義され、「品質」の構成要素と、それを測定・確認する方法を分析します。

開発環境

ライフサイクルと品質保証の要求事項を満たすために、コードは通常いくつかの異なる環境を通過します。

- 開発—コードはセキュアなサーバー上にホストされます。それぞれの開発者はコードの一部をチェックアウトし、自分のローカルマシンで編集します。通常、そうしたローカルマシンはサンドボックスとともに構成され、ローカルテストを行います。それにより、ローカル上で実行されている他のあらゆるプロセスが、開発中のアプリケーションに干渉したり、それを侵害したりしないことが保証されます。
- テスト/統合—この環境では、複数の開発者によるコードが単一のマスターコピーに統合され、(自動的に、もしくは人間のテスターによって) 基本的なユニットテストと機能テストを受けます。これらのテストは、コードが正しく構築され、設計が求める機能を満たしていることを保証するために行われます。
- ステージング**—これは本番環境と同じですが、テストデータやサンプルデータを使用することがあり、テストユーザーだけがアクセスできるよう追加のアクセス制御を備えています。この段階のテストは、利便性とパフォーマンスにより焦点を当てています。
- 本番—アプリケーションがエンドユーザーにリリースされます。



セキュアな開発環境。(画像提供 : © 123RF.com)

それぞれのコーディング環境の完全性を実証できることが重要です。どの環境における侵害も、侵害されたコードのリリースにつながる可能性があります。

- サンドボックス—各開発環境は、他の環境から分離されなければなりません。プロセスは、サンドボックスの外部に接続できないようにする必要があります。各サンドボックスには、コードの開発とテストを行うのに必要な最低限のツールとサービスのみ許可します。
- セキュアな構成ベースライン—各開発環境は、自動プロビジョニングなどを使用して、同じ仕様に構築される必要があります。
- 完全性の測定—このプロセスは、開発環境が構成ベースラインと異なっているかどうかを判断します。何らかのプログラミング問題を解決するために、開発者が不正なツールを追加したかもしれません。整合性の測定は、署名されていないファイルや、ベースラインとマッチしないファイルをスキャンすることで実行します。ファイル構造の比較には、Linuxの`diff`コマンドが用いられます(linux.die.net/man/1/diff)。

プロビジョニング、デプロビジョニング、およびバージョン管理

開発ライフサイクルモデルとQAプロセスを使用することで、過去の開発とテストを、アプリケーションもしくはスクリプトベースの自動化タスクのデプロイと保守に拡張することができます。

プロビジョニング

プロビジョニングは、企業のデスクトップ、モバイルデバイス、またはクラウド環境など、対象となる環境にアプリケーションをデプロイ（展開）することです。エンタープライズ版のプロビジョニングマネージャーは、複数のアプリケーションを1つのパッケージにまとめる場合もあります。それとは別に、OSとアプリケーションが単一のインスタンスとして定義され、仮想化プラットフォームでデプロイされることもあります。パッケージやインスタンスが最新のバージョンに更新されるよう、プロビジョニングプロセスでは、これらのアプリケーションの変更内容を考慮する必要があります。

デプロビジョニング

デプロビジョニングは、パッケージまたはインスタンスからアプリケーションを削除するプロセスです。ソフトウェアを完全に書き換える必要がある場合や、ソフトウェアがその目的をもはや満たさない場合に、デプロビジョニングを行います。アプリケーション自体を削除するのと同時に、環境を適切に変更して、そのアプリケーションをサポートするためだけの構成（ファイアウォールポートの開放など）を削除することが大切です。

バージョン管理

バージョン管理は、ソフトウェア製品の各バージョンのIDシステムです。バージョン管理番号のほとんどは、カスタマーもしくはエンドユーザーに通知されているバージョンと、開発プロセスで用いる内部ビルド番号の両方を表しています。バージョン管理は、ソフトウェア開発プロジェクトの変更管理プロセスをサポートします。ほとんどのソフトウェア開発環境ではビルドサーバーが用いられ、以前のバーションのソースコードのリポジトリを維持しています。開発者が新しいコードや変更されたコードをリポジトリに加えると、新しいコードは更新されたバージョン番号でタグ付けされ、古いバージョンはアーカイブされます。これにより、問題が判明した場合に元のバージョンに戻すことができます。

自動化/スクリプトリリースのパラダイム

コーディングプロジェクトは、さまざまなライフサイクルモデルを用いて管理されます。ウォーターフォールモデルのソフトウェア開発ライフサイクル(SDLC)は、ステージからステージへと進行する一本道なプロジェクトの成功に焦点を当てた古いパラダイムです。より最近のアジャイルの考え方には反復プロセスを使用し、十分にテストされたコードをより小さなブロックやユニットでリリースします。このモデルでは、開発およびプロビジョニングタスクは継続的なものと捉えられます。

継続的インテグレーション

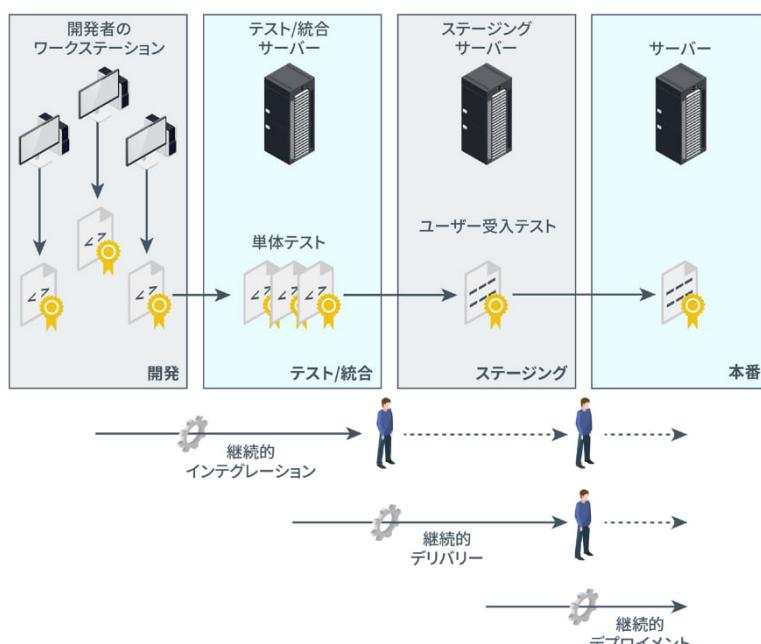
継続的インテグレーション(CI)とは、開発者は頻繁に（毎日もしくはそれ以上）更新を行い、それをテストすべきであるという原則です。これは、2人の開発者がコードの変更に時間を費やして、後で互いに衝突することが判明する可能性を減らすために設計されています。CIの狙いは、10件の競合の原因を診断するよりも、1件か2件の競合もしくはビルドエラーを診断する方が容易なため、早期のうちに検出して解決することです。効果的なCIを行うには、自動化されたテストスイートを用いてそれぞれのビルドを検証することが大切です。

継続的デリバリー

CIが開発におけるコード管理に関するものである一方、継続的デリバリーは、ネットワーキング、データベースの機能性、クライアントソフトウェアなど、そのアプリをサポートするすべてのインフラストラクチャのテストに関係しています。

継続的デプロイメント

継続的デリバリーは、あるバージョンのアプリケーションと、それをサポートするインフラストラクチャが本番環境に対応しているかどうかをテストしますが、継続的デプロイメントは、本番環境に実際に変更を加え、アプリの新しいバージョンをサポートする個別のプロセスです。



自動化および継続的リリースのパラダイム。(画像提供：© 123RF.com)

継続的モニタリングと自動化された一連のアクション

自動化ソリューションにはサービス障害やセキュリティインシデントを検知するための継続的モニタリングのシステムがあります。継続的モニタリングは、ローカルにインストールされたエージェントやハートビートプロトコルの使用や、可用性のチェックをリモートで行います。プライマリサイトの監視に加え、フェイルオーバーコンポーネントを観察し、すぐに復旧できるようにする確認も重要です。また、疑わしいと見なされるトラフィックを自動でブロックするようIPSを構成するなど、監視システムが取る基本方針を自動化することもできます。この種の機能は、SOAR (Security Orchestration, Automation, and Response) 管理ソフトウェアによって提供されます。

継続的バリデーション

アプリケーションモデルは、ソフトウェア開発プロジェクトを推進する要求事項のひとつです。この要求事項モデルは、確認・検証 (verification and validation、V&V) プロセスを用いてテストされます。

- 確認(verification)は、製品またはシステムがその設計目標を満たしていることを確認するための、コンプライアンステストプロセスです。
- 検証(validation)は、そのアプリケーションが目的に適合しているかどうか（設計目標がユーザーの要求事項を満たしているかどうか、など）を判断するプロセスです。

継続的パラダイムにおいて、設計目標がユーザーとセキュリティの要求事項を引き続き満たしているかどうか、デリバリーとデプロイのフィードバックを監視し評価する必要があります。モニタリングと検証のプロセスでは、セキュアな構成ベースラインからの逸脱がないことも確認しなければなりません。

ソフトウェアの多様性

アプリケーションのランタイム環境は、ホストシステム上での実行に関する次の2つのアプローチのどちらかを使用します。

- コンパイルされたコードが、対象のOS上で独立して動作することが可能な、バイナリマシン言語に変換される。
- 解釈されたコードはほぼ現状のままパッケージされるものの、PowerShellやJavaScriptなどのインタプリタによって1行ごとにコンパイルされる。インタプリタがOSの種類やバージョン間の違いを解決するので、プラットフォームに依存しないソリューションが提供される。

ソフトウェアの多様性が、悪意のあるコードを検知するのを難しくする、難読化テクニックを指すこともあります。これは、定評のあるShikata Ga Nai (fireeye.com/blog/threat-research/2019/10/shikata-ga-nai-encoder-still-going-strong.html)などのシグネチャ検知を避けるために、シェルコードコンパイラの形で、脅威アクターによって広く使用されています。これを防御テクニックとして使用することもできます。APIメソッドと自動化コードを難読化することで、脅威アクターがコードをリバースエンジニアリングして分析し、脆弱性を発見するのが難しくなります。

また、多様性によるセキュリティに関する一般的な研究にも関心が集まっています。これは、標準的でない環境に対して攻撃方法を開発することが難しいという原則の上に立っています。Windowsドメインネットワークなどの単一環境は、設定ミスを悪用するために利用可能な便利なマルウェアツールが多数あり、かなり予測可能な攻撃対象領域を提供します。さまざまな開発ツール、およびOS/アプリケーションのベンダーとバージョンを使用すると、攻撃戦略の調査が難しくなる可能性があります。これは、隠ぺいによるセキュリティと同じく、標的型攻撃を防ぐことはできませんが、より簡単なターゲットへと移っていく意欲の低い脅威アクターによるリスクを部分的に軽減できます。一方、この種の複雑性は、技術者や開発者が馴染みのない技術を習得するために苦労するために、設定エラーの発生率を高める傾向があります。

レビュー・アクティビティ：

デプロイと自動化のコンセプト

次の質問にお答えください。

1. セキュアなステージングとは何ですか？
2. プロビジョニングおよびデプロビジョニングのプロセスにおいてコードのバージョンを管理するには、どのような機能が不可欠ですか？
3. 本番環境に対するコードの継続的リリースを管理するライフサイクルプロセスは何ですか？
4. 特別に構成されたコンパイラは、ソフトウェアの多様性を通じてどのように攻撃を阻止しますか？

レッスン14

概要

アプリケーション攻撃を識別・分類して、開発とコーディングのベストプラクティスを要約できる必要があります。

セキュアなアプリケーション開発のガイドライン

アプリケーション開発プロジェクトを開始する、または改善させる際には、次のガイドラインに従ってください。

- セキュアなコーディング技法に関するトレーニングを開発者に施し、以下の攻撃に対する具体的な軽減策を伝える。
 - 脆弱なコードを悪用するオーバーフロー、競合状態、DLL/ドライバー操作攻撃。
 - 入力検証の欠如を悪用するインジェクション攻撃（XSS、SQL、XML、LDAP、シェルコード）。
 - セキュアな認証および認可メカニズムの欠如を悪用する、リプレイ攻撃およびリクエストフォージェリ攻撃
- 静的・動的分析を使用してコードのレビューとテストを行い、特に入力検証、出力エンコーディング、エラー処理、およびデータの漏洩に注意を払う。
- 自動化と、継続的インテグレーション/デリバリー / デプロイ/監視/検証を活用し、セキュアかつ一貫した開発、ステージング、本番環境を確保する。
- 承認されたコーディング言語の使用と起動場所を文書化し、理想的にはコード署名を利用して、悪意のあるコードを容易に検知できるようにする。

レッスン15

セキュアなクラウドソリューションの実装

レッスン概要

クラウドコンピューティングの背後にある主要な考え方は、世界中のどこにいても、どのホストからでも、自分のデータとアプリケーションにアクセスしてそれらを管理できる一方、ストレージの方法と場所は仮想化を通じて隠されているか抽象化されている、ということです。クラウドアプリケーションは、パブリックサービスとしてアクセスする形であっても、あるいはプライベートな仮想化インフラストラクチャを介してプロビジョニングされる形であっても、オンプレミスのサービスデリバリー・モデルを急速に追い越しつつあります。クラウドのセキュリティ面における検討事項が、あなたのセキュリティ・プロフェッショナルとしてのキャリアにおいて今後ますます重要な部分になるでしょう。

レッスンの目的

このレッスンの内容は、以下のとおりです。

- セキュアなクラウドサービスと仮想化サービスを要約する。
- クラウドセキュリティソリューションを適用する。
- インフラストラクチャとしてのコードの概念として要約する。

トピック15A

セキュアなクラウドサービスと仮想化サービスを要約する



対象試験範囲

2.2仮想化とクラウドコンピューティングのコンセプトを要約することができる。

従来のインフラストラクチャにおいては、ネットワークが外部の世界から隔離されていることがあるため、脅威アクターによる侵入が困難な場合もあります。しかしクラウド環境では、脅威アクターはインターネット接続と、盗まれたパスワードハッシュかSSH鍵の辞書さえあれば侵害を引き起こせます。クラウドプロバイダーのセキュリティ手順が管理されていなければ、組織の背負うリスクが劇的に増加する可能性もあります。あなたはセキュリティプロフェッショナルとして、クラウドサービスとそのデリバリー・モデル、およびそれらを支える仮想化テクノロジーに関連する脅威と脆弱性を評価できなければなりません。

クラウドデプロイモデル

クラウドデプロイモデルは、サービスがどのように所有され、プロビジョニングされるかを分類するものです。各デプロイモデルが脅威や脆弱性に対して及ぼすさまざまな影響を認識することが重要になります。クラウドデプロイモデルは、広く次のように分類されます。

- **パブリック（またはマルチテナント）** – サービスはクラウドサービスプロバイダー (CSP)によって、インターネット経由でクラウドコンシューマーに提供されます。このモデルでは、企業はサブスクリプションまたは従量課金制を採用し、同時に下位のサービスを無料で提供することができます。共有リソースのため、パフォーマンスとセキュリティに関するリスクが存在します。マルチクラウドアーキテクチャでは、組織は複数のCSPが提供するサービスを使用します。
- ホステッドプライベート – その組織専用のクラウド環境として、サードパーティがホストします。これはより安全で、高いパフォーマンスが保証されますが、その分高額です。
- **プライベート** – 完全にその組織専用であり、その組織が所有しているクラウドインフラストラクチャです。このモデルでは、1つの事業部門がクラウド管理を専門に行なうことが一般的で、他のビジネスユニットがそれを利用します。プライベートクラウドコンピューティングにより、組織は自らのサービスのプライバシーとセキュリティをより自由に制御できます。このデプロイモデルは、特に運用に際して厳格なアクセス制御を必要とする銀行や政府のサービスに適しています。
- **コミュニティ** – ホステッドプライベートまたは完全にプライベートなクラウドのコストを複数の組織で分担するものです。一般的には、標準化やセキュリティポリシーといった共通の関心事に関するリソースをプールすることが目的です。

また、パブリック/プライベート/コミュニティ/ホスティッド/オンサイト/オフサイトといった各ソリューションのハイブリッドとして実装するクラウドコンピューティングソリューションもあります。たとえば、ある旅行会社で1年の大半はプライベートクラウドで販売用Webサイトを運営し、高い使用率が予想される時期だけパブリッククラウドにそのソリューションを分割するといったケースがこれに該当します。

柔軟性はクラウドコンピューティングの主要な利点ですが、プライベートストレージ環境とパブリックストレージ環境との間でデータを移動させる際には、データのセキュリティリスクをよく理解しておく必要があります。

クラウドサービスモデル

クラウドサービスは所有モデル（パブリック、プライベート、ハイブリッド、コミュニティ）だけでなく、複雑さと提供される事前設定サービスのレベルによっても分類することができます。これらのモデルは**Anything as a Service（エックスアズアサービス：XaaS）**と呼ばれます。最も一般的に実装されるのは、インフラストラクチャ、ソフトウェア、そしてプラットフォームの3つです。

Infrastructure as a Service

Infrastructure as a Service（サービスとしてのインフラストラクチャ：IaaS）は、サーバーやロードバランサー、ストレージエリアネットワーク(SAN)コンポーネントなどのITリソースを迅速にプロビジョニングする手段の1つです。ユーザーは、これらのコンポーネントやインターネットリンクを購入する代わりに、サービスプロバイダーのデータセンターから必要に応じてレンタルします。その例として、Amazon Elastic Compute Cloud (aws.amazon.com/ec2)、Microsoft Azure Virtual Machines (azure.microsoft.com/services/virtual-machines)、Oracle Cloud (oracle.com/cloud)、およびOpenStack (openstack.org)があります。

Software as a Service

Software as a Service（サービスとしてのソフトウェア：SaaS）は、ソフトウェアアプリケーションのプロビジョニングモデルの1つです。ユーザーはソフトウェアライセンスを一定のシート数で購入する代わりに、サプライヤーのサーバーでホストされているソフトウェアに従量課金制またはリース契約（オンデマンド）でアクセスすることになります。仮想インフラストラクチャを利用するため、開発者は従来よりも迅速にオンデマンドでアプリケーションをプロビジョニングすることができます。アプリケーションはクラウド内で開発・テストでき、クライアントコンピューター上でテストしてデプロイする必要がありません。例としてMicrosoft Office 365 (microsoft.com/en-us/microsoft-365/enterprise)、Salesforce (salesforce.com)、およびGoogle G Suite (gsuite.google.com)があります。

Platform as a Service

Platform as a Service（サービスとしてのプラットフォーム：PaaS）は、SaaSとIaaSの中間でリソースを提供します。一般的なPaaSソリューションは（IaaSと同様に）サーバーとストレージのネットワークインフラストラクチャを提供しますが、それだけでなく同時に多層Webアプリケーション/データベースプラットフォームも提供します。このプラットフォームは、OracleまたはMS SQLや、PHP、MySQLをベースとしている場合があります。例としてOracle Database (oracle.com/database)、Microsoft Azure SQL Database (azure.microsoft.com/services/sql-database)、およびGoogle App Engine (cloud.google.com/appengine)があります。

SaaSと大きく違うのは、このプラットフォームは実際に何かを実行するように構成されていない、という点です。そのため、社内の開発者がCRMやeコマースアプリケーションなど、そのプラットフォームで実行するソフトウェアを作成する必要があります。サービスプロバイダーは、プラットフォームコンポーネントの完全性と可用性を確保しますが、プラットフォーム上に作成したアプリケーションのセキュリティは各ユーザーの責任となります。

The screenshot shows the AWS EC2 Dashboard. On the left, there's a sidebar with navigation links for EC2 Dashboard, Events, Tags, Reports, Limits, Instances, Launch Templates, Spot Requests, Reserved Instances, Dedicated Hosts, Scheduled Instances, AMIs, Bundle Tasks, Volumes, Snapshots, and Network & Security. The main content area is titled 'Resources' and displays the following information:

- You are using the following Amazon EC2 resources in the US East (N. Virginia) region:
- 0 Running Instances**
- 0 Dedicated Hosts**
- 0 Volumes**
- 0 Key Pairs**
- 0 Placement Groups**
- 0 Elastic IPs**
- 0 Snapshots**
- 0 Load Balancers**
- 1 Security Groups**

A callout box says: "Learn more about the latest in AWS Compute from AWS re:Invent 2017 by viewing the [EC2 Videos](#).
x".

Below this, there's a 'Create Instance' section with a 'Launch Instance' button. A note says: "Note: Your instances will launch in the US East (N. Virginia) region".

On the right side, there's a 'Service Health' section and a 'Scheduled Events' section. At the bottom right, there's a 'AWS Marketplace' section with a note: "Find free software trial products in the AWS Marketplace from the [EC2 Launch Wizard](#). Or try these".

At the very bottom, there are links for Feedback, English (US), Copyright notice (© 2008 - 2018, Amazon Web Services, Inc. or its affiliates. All rights reserved.), Privacy Policy, and Terms of Use.

Amazon Web ServicesのElastic Compute Cloud (EC2) IaaS/PaaSのダッシュボード。

(スクリーンショットは[Amazon.com](#)からの許可を得て使用。)

Anything as a Service

XaaSの例は他にも多数あり、何であってもクラウドサービスとしてプロビジョニングできるという考え方を反映しています。たとえば、サービスとしてのデータベースやサービスとしてのネットワークは、サービスとしてのプラットフォームのさらに具体的なものとして区別することができます。これらのモデルに関するセキュリティ上の主要な検討事項は、責任がどこにあるかを特定することです。多くの場合、これは「クラウド内のセキュリティ」と「クラウドのセキュリティ」の違いと呼ばれます。クラウド内のセキュリティとは、あなたが責任を負わなければならない事柄であり、クラウドの責任はCSPが管理する事柄です。それらの責任はサービスの種類に応じて異なります。

責任	IaaS	PaaS	SaaS
IAM	あなた	あなた	あなた (CSPツールセットを使用)
データセキュリティ (CIA属性/バックアップ)	あなた	あなた	あなた/CSP/両者
データプライバシー	あなた/CSP/両者	あなた/CSP/両者	あなた/CSP/両者
アプリケーションコード/構成	あなた	あなた	CSP
仮想ネットワーク/ ファイアウォール	あなた	あなた/CSP	CSP
ミドルウェア (データベース) コード/構成	あなた	CSP	CSP
仮想ゲストOS	あなた	CSP	CSP
仮想化レイヤー	CSP	CSP	CSP
ハードウェアレイヤー (コンピューティング、ストレージ、ネットワーキング)	CSP	CSP	CSP



この責任分担表は一般的な責任しか示していないことに注意してください。具体的な規定は、CSPとの契約書やサービスレベル同意書 (Service Level Agreement : SLA) の中に定める必要があります。

Security as a Service

さまざまなテクノロジーにおいてセキュリティに関する専門家レベルの知識と構成が必要とされおり、企業がどこかの時点でサードパーティのサポートに頼らざるを得ない可能性が高くなっています。そうしたサポートは、次の3つの「層」に分類することができます。

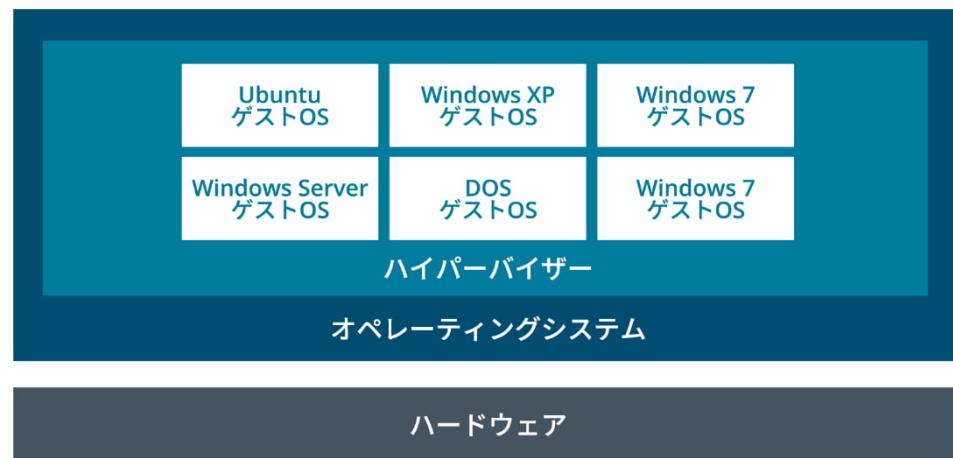
- **コンサルタント**（大小問わず）どのようなタイプの組織においても、サードパーティの専門家が持つ経験と視点は、セキュリティに関するその組織の意識と能力を向上させる上で大いに有益です。コンサルタントはフレームワークの「全体像」を分析・調整するために活用されることもあるれば、特定のプロジェクトや製品に焦点を当てたプロジェクト（ペントスト、SIEMのロールアウトなど）のために活用されることもあります。また、機能を実装するためなく、能力を開発するためにコンサルタントを活用するのであれば、コストをコントロールするのも極めて容易です。コンサルタントがセキュリティ機能を「所有」するようになると、関係を変えたり断ち切ったりするのが難しくなることもあります。
- **マネージドセキュリティサービスプロバイダー (MSSP)** – 情報保証の責任をサードパーティに完全に外部委託する手段です。このタイプのソリューションは高額ですが、急速に成長したため社内にセキュリティ能力がない中小企業にとって優れた選択肢となり得ます。もちろん、この種の外部委託はMSSPに大きな信頼を置くことになります。MSSPを効果的に管理し続けるには、セキュリティについてかなりの意識と技能が組織内に必要です。また、情報処理に関して高度な規制を課せられている業界では、かなりの課題が伴うこともあります。
- **Security as a Service (サービスとしてのセキュリティ : SECaS)** – これは多数の異なる事柄を意味しますが、通常はウイルススキャンやSIEMに似た機能など、クラウド内に特定のセキュリティ管理を実装する手段として、MSSPからは区別されます。一般的に、クラウドサービスへのコネクターがローカル環境でインストールされることになります。一例を挙げると、アンチウイルスエージェントはローカル環境でファイルをスキャンしますが、管理と更新はクラウドプロバイダーから行われます。同様に、ログコレクターはクラウドサービスにイベントを送信し、そこで集約と相互関連が行われます。例としてCloudflare (cloudflare.com/saas)、Mandiant/FireEye (fireeye.com/mandiant/managed-detection-and-response.html)、およびSonicWall (sonicwall.com/solutions/service-provider/security-as-a-service)があります。

仮想化テクノロジーとハイバーバイザーの各タイプ

仮想化は、1台のコンピューターに複数のオペレーティングシステムがインストールされ、同時に実行できることを意味します。仮想プラットフォームには、少なくとも以下の3つのコンポーネントが必要です。

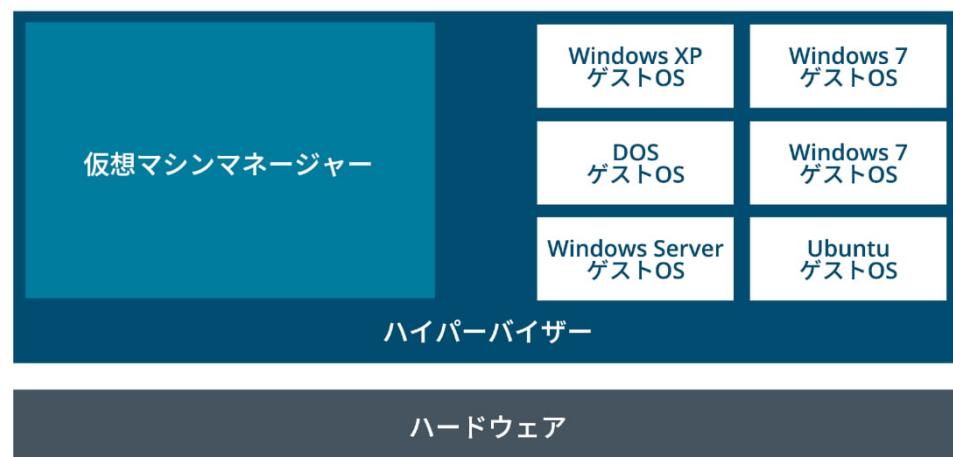
- **ホストハードウェア** – 仮想環境をホストするプラットフォームです。また、複数のホストをネットワークでつなげるという選択肢もあります。
- **ハイバーバイザー /仮想マシンモニター (VMM)** – 仮想マシン環境を管理し、コンピューター・ハードウェアとネットワークの相互作用を円滑にします。
- **ゲストオペレーティングシステム、仮想マシン(VM)**、またはインスタンス – 仮想環境下でインストールされたオペレーティングシステムです。

仮想プラットフォームの基本的な違いは、ホストハードウェアとの相互作用の方式がホスト型であるか、ベアメタル型であるかです。ゲストOS（またはホスト型）システムでは、Type IIハイバーバイザーと呼ばれるハイバーバイザーアプリケーション自体がホストのオペレーティングシステムにインストールされています。ホスト型ハイバーバイザーの例としては、VMware Workstation、Oracle VirtualBox、およびParallels Workstationなどがあります。ハイバーバイザーソフトウェアは、ホストOSをサポートしている必要があります。



ゲストOSの仮想化 (Type II/ハイパー・バイザ) — ハイパー・バイザはネイティブOS内で実行されるアプリケーションで、ゲストOSはハイパー・バイザ内にインストールされます。

ベアメタル型の仮想化プラットフォームでは、Type I/ハイパー・バイザと呼ばれるハイパー・バイザがコンピューターに直接インストールされており、ホストのOSを経由せずにホストハードウェアへのアクセスを管理します。例としてVMware ESXi Server、MicrosoftのHyper-V、CitrixのXEN Serverなどがあります。ハードウェアはハイパー・バイザの基本システム要件と、インストールするゲストOSの種類と数に応じたリソースだけはサポートする必要があります。



Type Iの「ベアメタル」型ハイパー・バイザ — ハイパー・バイザは管理アプリケーションと共にホストハードウェアに直接インストールされ、次いでVMがハイパー・バイザ内にインストールされます。

仮想デスクトップインフラストラクチャとシンクライアント

仮想デスクトップインフラストラクチャ (VDI)は、VMを使用して企業のデスクトップをプロビジョニングする手法です。典型的なVDIでは、デスクトップコンピューターの代わりに低スペック・省電力のシンクライアント(thin client)コンピューターが使用されます。シンクライアントを立ち上げると、最低限のOSが起動し、ユーザーは会社のサーバーインフラストラクチャに保存されたVMにログオンできます。ユーザーはリモートデスクトッププロトコル (Microsoft Remote Desktop

またはCitrix ICAなど) を使ってVMに接続します。シンクライアントは正しいイメージを見つけ、適切な認証メカニズムを使用する必要があります。マシン名またはIPアドレスに基づく1:1マッピングがあることがあります。またイメージを見つけるためのプロセスを接続ブローカーが行うことがあります。

仮想デスクトップ環境(VDE)またはワークスペースにおいて、アプリケーションの処理とデータのストレージは、すべてサーバーにより実行されます。シンクライアントコンピューターは、画面イメージの表示とオーディオの再生を行い、マウス、キーボード、ビデオ、オーディオの各情報をネットワーク経由で転送できるだけの処理能力があれば十分です。すべてのデータはサーバーに保存されるため、バックアップが簡単に行えます。また、デスクトップVMではサポートとトラブルシューティングも容易に行えます。ただし、VMで行った変更はテンプレートイメージから簡単に上書きできるため、安全でないユーザー操作は「ロック」することが推奨されます。VDIを利用することで、企業はITインフラストラクチャの運用をサードパーティサービス会社に完全に委託することができます。

主なデメリットとしては、サーバーやネットワークインフラストラクチャに障害が発生した場合に、ユーザーにローカル処理を実行する能力がないので、生産性の喪失という点で稼働停止のコストがより高額になるという点が挙げられます。

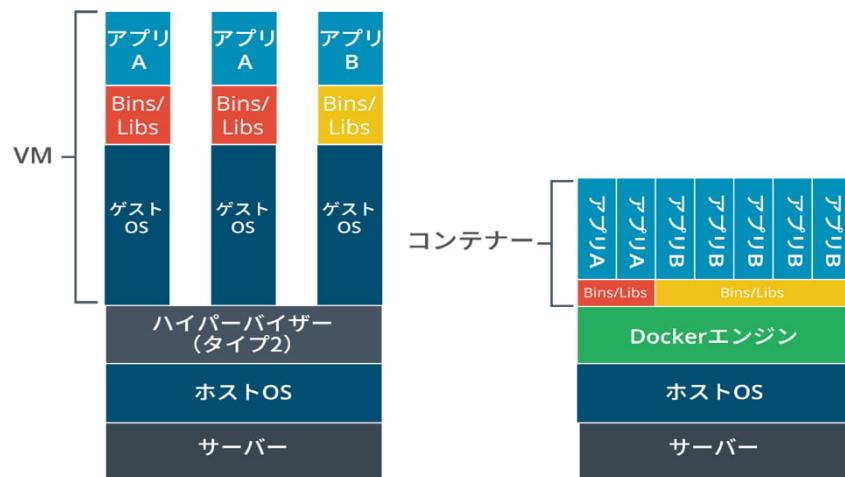
アプリケーションの仮想化とコンテナー型仮想化

アプリケーションの仮想化は、より限定的なVDIの一種です。クライアントデスクトップ全体を仮想プラットフォームとして動作させる代わりに、クライアントはサーバー上でホストされているアプリケーションにアクセスするか、ローカル処理を行えるようにサーバーからクライアントにアプリケーションをストリームします。大半のアプリケーション仮想化ソリューションはCitrix XenApp (旧称MetaFrame/Presentation Server) をベースと/orしていますが、Microsoftは自社のWindows Server製品群と共にApp-V製品を開発しており、VMwareもThinApp製品を用意しています。現在では多くの場合、これらのタイプのソリューションはHTML5リモートデスクトップアプリと共に使用されており、ユーザーは通常のWebブラウザーソフトウェア経由でそれらにアクセスできるので、「クライアントレス」と呼ばれています。

アプリケーションのセル型/コンテナー型仮想化は、ハイパーテーブイザーを使用せず、オペレーティングシステムレベルでのリソースの分離に主眼を置いた仮想化モデルです。OSは、各ユーザーインスタンスで実行先となる分離された「セル」を定義します。それぞれのセルないしコンテナーにはCPUとメモリリソースが割り当てられますが、すべてのプロセスはネイティブOSカーネルを介して実行されます。これらのコンテナーは多少異なるOSディストリビューションを実行できますが、異なる種類のゲストOSを実行することはできません (たとえば、Red Hat LinuxコンテナーでWindowsやUbuntuを実行するなど)。また、コンテナーは個別のアプリケーションプロセスを実行することができますが、その場合にはアプリケーションプロセスで必要な変数とライブラリがコンテナーに追加されます。

最も有名なコンテナー型仮想化製品はDocker ([docker.com](https://www.docker.com))です。コンテナー化は多数のクラウドサービスを支えており、特にマイクロサービスとサーバーレスアーキテクチャをサポートしています。企業のワークスペースをモバイルデバイスに実装する手段としても、コンテナー化が広く利用されています。

コンテナーとVM



VMとコンテナーの比較。

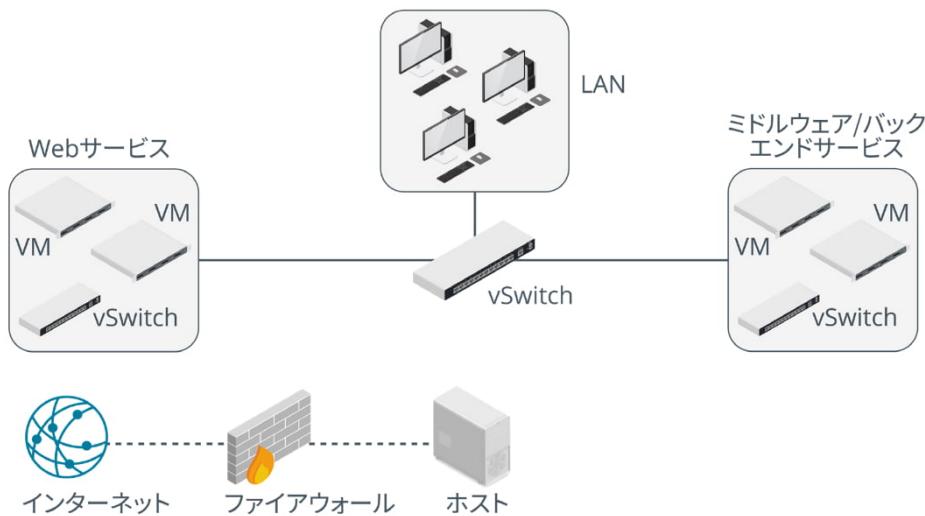
VMエスケープ保護

VMエスケープは、ゲストOSで実行されているマルウェアが別のゲストやホストに移動することを指します。それを行うために、マルウェアは自身が仮想環境で実行されていることを認識しなければなりませんが、それは通常簡単に行えます。その手段の1つにタイミング攻撃があります。古典的なタイミング攻撃では、複数のユーザー名を認証サーバーに送り、そのサーバーの反応時間を測定することになります。通常、無効なユーザー名はすぐさま拒否されますが、有効なユーザー名はより長い時間がかかります（その間、認証サーバーはパスワードをチェックしています）。これにより、脅威アクターは有効なユーザー名を入手できます。マルウェアはゲストOS内でタイミング攻撃を使用し、それがVMで動作しているかどうかを検知します（「現実」環境と比べた場合、特定の操作にかかる時間が違う場合があります）。脅威アクターが仮想システムハードウェアの存在を突き止めるために用いる「シグネチャ」は、他にも多数存在します。VMエスケープの次なる段階は、脅威アクターによるハイパーバイザの侵害です。セキュリティの研究者はこの種のエクスプロイトに焦点を当てており、いくつかの脆弱性が人気のあるハイパーバイザから見つかっています。

ホストされたアプリケーションに仮想化が用いられている場合、VMエスケープは深刻な意味を持ちます。ホストされたWebサーバーを運用している場合、ホスティングプロバイダーに自分のデータを委ねることはさておき、他のカスタマーのVMで他にどのようなアプリケーションが実行されているか、あなたにはまったくわかりません。例として、自分のEコマースWebサーバーを、あるISPからリースされた仮想サーバー上にインストールしているというシナリオを検討しましょう。サードパーティが別のゲストOSをインストールし、そこにその仮想サーバーのハイパーバイザを侵害し得るマルウェアが含まれている場合、そのサードパーティはあなたのサーバーに、または物理サーバーのメモリーに保持されているデータにアクセスできるでしょう。ハイパーバイザを侵害したサードパーティは、あなたのサーバーイメージのコピーを作成し、それをどの場所にでもダウンロードできるでしょう。それにより、脅威アクターはEコマースサーバー上で保持されている暗号化されていないデータをすべて盗み出せるでしょう。さらに悪いことに、サーバーに保存されている秘密の暗号鍵を入手したり、暗号化されていないデータや物理サーバーのメモリにあるデータ暗号鍵を盗み出したりすることで、暗号化されたデータを盗むことができる可能性も考えられます。

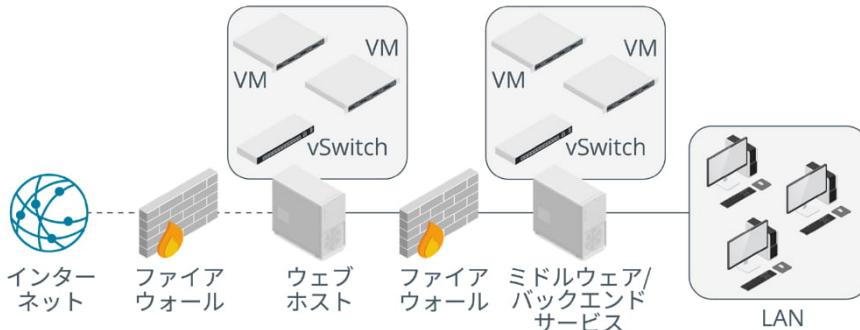
セキュリティ関連の刊行物をチェックして、自分が運用しているハイパーバイザーソフトウェアの記事を調べると共に、パッチと更新を迅速にインストールすることが不可欠です。また、異なるセキュリティ要件を有するさまざまなタイプのアプリケーションを実行するVMを設置することで不要なリスクが生じないよう、VMアーキテクチャを慎重に設計する必要があります。

VMエスケープの防止は、仮想化ベンダーがハイパー・バイザにおけるセキュリティの脆弱性を特定すること、およびそれらにパッチが適用されることに依存しています。VMをデプロイする際に効果的なサービス設計とネットワーク配置を行なうことで、VMエスケープの影響を軽くすることができます。



ゾーンを仮想化されたデバイスに落とし込む – この構成はVMエスケープ攻撃に対して極めて脆弱です。
(画像提供: © 123RF.com)

たとえば、DMZなどのセキュリティゾーンを検討する際は、フロントエンドサービスを提供するVMと、ミドルウェア/バックエンドサービスを提供するVMは、別々の物理ホストに分離しなければなりません。これにより、DMZ内のホストに対するVMエスケープ攻撃のセキュリティ面での影響が軽くなります（通常、DMZのホストはこうした攻撃に対してより脆弱です）。



VMを個別のハードウェア上の異なるゾーンに隔離する – これにより、VMエスケープ攻撃の衝撃が減少するはずです。
(画像提供: © 123RF.com)

VMスプロールの回避

ハイパー・バイザを保護することに加え、1つ1つのVMを他のあらゆるネットワークホストと同じように扱うことも必要です。これは、セキュリティポリシーとセキュリティ管理を用いることで、ホストの仮想化に依存しているすべてのデータとサービスの機密性、完全性、可用性を保証することを意味します。

各VMはそれぞれのセキュリティソフトウェアスイートと共にインストールし、マルウェアや侵入の試みに対して保護する必要があります。また各ゲストにもパッチ管理プロセスを実施しなければなりません。これは、ローカル環境に更新をインストールするか、更新されたVMテンプレートイメージでゲストインスタンスを置き換えることを意味します。



ホストにインストールされた通常のウイルス対策ソフトウェアでは、ゲスト OS に侵入したウイルスを検出することはできません。ゲスト OS の仮想ディスクをホストからスキャンすると、重大なパフォーマンスの問題が発生する可能性があります。

仮想化の主な利点の1つとして、新しいシステムを容易にデプロイできることが挙げられます。が、この種のシステムスプロールと文書化されていないアセットのデプロイが、セキュリティ問題の根源になることもあります。何かをテストするためにシステムを「短時間だけ」立ち上げたものの、それが文書化も保護もパッチの適用もされないまま、数カ月ないし数年にわたって放置されることがよくあります。そうした文書化されていないシステムの1つ1つが、悪用可能な脆弱性を表すこともあります。それにより、ネットワークの潜在的な攻撃対象領域が広がります。仮想化されたアセットの追跡、保護、および破棄（すでに使用されていない場合）のポリシーと手順を定め、慎重に実施する必要があります。

仮想マシンライフサイクル管理(VMLM)ソフトウェアをデプロイすることで、**VMスプロール**の回避を実行できます。VMLMソリューションにより、組織内のすべての仮想環境のメンテナンスと監視を行う中央化されたダッシュボードが得られます。より一般的には、マシンイメージの開発と展開における管理手順を厳密に策定し、監視する必要があります。VMはアプリケーション固有のテンプレートに従っており、最小限の構成で（不要なサービスを実行することなく）そのアプリケーションを実行できる必要があります。また、マルウェアに感染する可能性がある環境や、悪意のあるコードが挿入されている可能性がある環境で、イメージを実行すべきではありません。ここでの最大の懸念は、不正な開発者や請負業者によってマシンのイメージ内にバックドアや「論理爆弾」を仕込まれることです。犯罪者や職場に不満を持つスタッフがセキュリティ環境に悪影響を及ぼすことは明らかですが、VMマシンイメージの中にコードを隠すことは簡単であり、はるかに深刻な被害をもたらす可能性があります。

レビュー アク ティビティ：

セキュアなクラウドサービス と仮想化サービス

次の質問にお答えください。

1. パブリッククラウドは何を意味しますか？
2. SANの実装にはどのタイプのクラウドソリューションが用いられますか？
3. Type IIハイパーバイザーとは何ですか？
4. VDEとは何ですか？
5. VMエスケープ攻撃から生じるリスクはどういったものですか？

トピック15B

クラウドセキュリティソリューションを適用する



対象試験範囲

- 1.2 与えられたシナリオに基づいて、可能性あるインジケーターを分析して攻撃のタイプを特定することができる。(クラウドベースとオンプレミスの違いのみ)
- 2.2 仮想化とクラウドコンピューティングのコンセプトを要約することができる。
- 3.6 与えられたシナリオに基づいて、クラウドにサイバーセキュリティソリューションを適用することができる。

クラウドセキュリティソリューションの構成では、多数の原則やプロセスがオンプレミスセキュリティと共に通っていますが、馴染みのないテクノロジーや課題も数多くあります。クラウドサービスの脆弱な構成によって数多くの攻撃ベクターが利用可能になり、またクラウドが持つパブリックな性質のために、それらがすぐさま発見され悪用されることになります。機密性、完全性、可用性の各属性を持つ、コンピューティング、ネットワーク、ストレージクラウドのリソースのプロビジョニングに向け、技術的制御のポリシーを適用できるようになる必要があります。

クラウドセキュリティの統合と監査

クラウド型のサービスは通常のセキュリティポリシーとセキュリティ手順の中に統合され、コンプライアンスの監査を受けなければなりません。オンプレミス攻撃のインジケーターがローカルアプリケーションのログやネットワークトラフィックから発見される一方、クラウド型の攻撃のインジケーターはAPIのログや測定値から発見されます。疑わしいIPアドレス範囲やドメインと、疑わしいコード文字列との間には、同じ相関関係があるはずですが、このデータのソースはクラウドサービスプロバイダー(CSP)となります。この監査情報をリアルタイムで評価することは、クラウドサービスのタイプによっては難しいことがあります。オンプレミスとクラウド型両方のネットワークおよびインスタンスから得られたセキュリティデータの収集、集約、そして相互の関連付けを行える、クラウド型のSIEMソリューションが多数存在しています。

あらゆる契約型サービスと同じく、クラウドコンピューティングはリスクを移転する手段となります。したがって、自分が移転しているのはどのリスクかを正確に特定し、サービスプロバイダーが負っている責任はどれか、自分に残っている責任はどれかを識別することが不可欠です。これは責任マトリックスとしてサービスレベル同意書 (Service Level Agreement : SLA) に明記しなければなりません。たとえば、あるSaaSソリューションにおいて、プロバイダーがソフトウェアの機密性、完全性、可用性に責任を負っているとします。するとそのプロバイダーは、耐障害性とクラスタ化されたサーバーサービスの構成、サーバーへのファイアウォール設置と適切な認証、認可、アカウンティング手順の作成、侵入のスキャンとネットワークログの監視、OSとソフトウェアのパッチの適用などに責任を負うことになるでしょう。あなたは、ソフトウェア管理機能の一部または全部に責任を負うことがあるかもしれません、管理者とユーザーによる良好なパスワード管理、システム権限の設定、データのバックアップ作成などには責任があります。

重要なタスクがサービスプロバイダーの責任であっても、そうしたタスクが完了したことを示す報告メカニズムがある、またはサービスプロバイダーの災害復旧計画が効果的であるといったことを確認しなければなりません。

別の条件として、深刻なセキュリティ違反に対してはあなたの会社が依然として責任を負う可能性がある、というものがあります。たとえば顧客データが盗まれた場合、あるいは自社のホストされたWebサイトがハッキングされ、マルウェアの配布に用いられた場合がそれに該当します。また法令や規制による要求事項についても責任を負っています。損害についてサービスプ