DevSecOpsの6つの柱：
自動化

SAFECode
Software Assurance Forum for Excellence in Code
Driving Security and Integrity

CSA cloud security alliance®

DevSecOpsワーキンググループの恒久的かつ公式な場所は、
https://cloudsecurityalliance.org/research/working-groups/devsecops/です。

Acknowledgments

Lead Authors:

Souheil Moghnie
Theodore Niedzialkowski
Sam Sehgal

Contributor:

Michael Roza

CSA Analysts:

Sean Heide

Special Thanks:

Ankur Gargi
Raj Handa
Manuel Ifland
John Martin
Kamran Sadique
Charanjeet Singh
Altaz Valani

日本語版提供に際しての告知及び注意事項

本書「DevSecOpsの6つの柱：自動化」は、Cloud Security Alliance (CSA)が公開している「The Six Pillars of DevSecOps: Automation」の日本語訳です。本書は、CSAジャパンが、CSAの許可を得て翻訳し、公開するものです。原文と日本語版の内容に相違があった場合には、原文が優先されます。
翻訳に際しては、原文の意味および意図するところを、極力正確に日本語で表すことを心がけていますが、翻訳の正確性および原文への忠実性について、CSAジャパンは何らの保証をするものではありません。
この翻訳版は予告なく変更される場合があります。以下の変更履歴（日付、バージョン、変更内容）をご確認ください。

変更履歴

日付	バージョン	変更内容
2023年4月3日	日本語版1.0	初版発行

CSAジャパン成果物の提供に際しての制限事項

日本クラウドセキュリティアライアンス（CSAジャパン）は、本書の提供に際し、以下のことをお断りし、またお願いします。以下の内容に同意いただけない場合、本書の閲覧および利用をお断りします。

1.　責任の限定
CSAジャパンおよび本書の執筆・作成・講義その他による提供に関わった主体は、本書に関して、以下のことに対する責任を負いません。また、以下のことに起因するいかなる直接・間接の損害に対しても、一切の対応、是正、支払、賠償の責めを負いません。
(1)　本書の内容の真正性、正確性、無誤謬性
(2)　本書の内容が第三者の権利に抵触しもしくは権利を侵害していないこと
(3)　本書の内容に基づいて行われた判断や行為がもたらす結果
(4)　本書で引用、参照、紹介された第三者の文献等の適切性、真正性、正確性、無誤謬性および他者権利の侵害の可能性

2.　二次譲渡の制限
本書は、利用者がもっぱら自らの用のために利用するものとし、第三者へのいかなる方法による提供も、行わないものとします。他者との共有が可能な場所に本書やそのコピーを置くこと、利用者以外のものに送付・送信・提供を行うことは禁止されます。また本書を、営利・非営利を問わず、事業活動の材料または資料として、そのまま直接利用することはお断りします。
ただし、以下の場合は本項の例外とします。
(1)　本書の一部を、著作物の利用における「引用」の形で引用すること。この場合、出典を明記してください。
(2)　本書を、企業、団体その他の組織が利用する場合は、その利用に必要な範囲内で、自組織内に限定して利用すること。

(3)　CSAジャパンの書面による許可を得て、事業活動に使用すること。この許可は、文書単位で得るものとします。
(4)　転載、再掲、複製の作成と配布等について、CSAジャパンの書面による許可・承認を得た場合。この許可・承認は、原則として文書単位で得るものとします。

3.　本書の適切な管理
(1)　本書を入手した者は、それを適切に管理し、第三者による不正アクセス、不正利用から保護するために必要かつ適切な措置を講じるものとします。
(2)　本書を入手し利用する企業、団体その他の組織は、本書の管理責任者を定め、この確認事項を順守させるものとします。また、当該責任者は、本書の電子ファイルを適切に管理し、その複製の散逸を防ぎ、指定された利用条件を遵守する（組織内の利用者に順守させることを含む）ようにしなければなりません。
(3)　本書をダウンロードした者は、CSAジャパンからの文書（電子メールを含む）による要求があった場合には、そのダウンロードしまたは複製した本書のファイルのすべてを消去し、削除し、再生や復元ができない状態にするものとします。この要求は理由によりまたは理由なく行われることがあり、この要求を受けた者は、それを拒否できないものとします。
(4)　本書を印刷した者は、CSAジャパンからの文書（電子メールを含む）による要求があった場合には、その印刷物のすべてについて、シュレッダーその他の方法により、再利用不可能な形で処分するものとします。

4.　原典がある場合の制限事項等
本書がCloud Security Alliance, Inc.の著作物等の翻訳である場合には、原典に明記された制限事項、免責事項は、英語その他の言語で表記されている場合も含め、すべてここに記載の制限事項に優先して適用されます。

5.　その他
その他、本書の利用等について本書の他の場所に記載された条件、制限事項および免責事項は、すべてここに記載の制限事項と並行して順守されるべきものとします。本書およびこの制限事項に記載のないことで、本書の利用に関して疑義が生じた場合は、CSAジャパンと利用者は誠意をもって話し合いの上、解決を図るものとします。

その他本件に関するお問合せは、info@cloudsecurityalliance.jp までお願いします。

日本語版作成に際しての謝辞

「DevSecOpsの6つの柱：自動化」は、CSAジャパン会員の有志により行われました。作業は全て、個人の無償の貢献としての私的労力提供により行われました。なお、企業会員からの参加者の貢献には、会員企業としての貢献も与っていることを付記いたします。
以下に、翻訳に参加された方々の氏名を記します。（氏名あいうえお順・敬称略）

高橋 久緒, CISSP, RISS, PMP
松浦 一郎, CISSP, CISM, CDPSE
満田 淳

目次

序文

クラウドセキュリティアライアンスとSAFECodeは、ソフトウェアセキュリティの成果を向上させることに深くコミットしています。2019年8月に発表された論文「*Six Pillars of DevSecOps*」は、ソフトウェアを、スピード感を持ちながらセキュリティ関連のバグを最小限に抑えて構築するために著者が用いた手法および実装に成功したソリューションに関するハイレベルなまとめを提供するものです。その6つの柱とは：

- 柱1：集団的責任（2020/02/20掲載）
- 柱2：トレーニングおよびプロセスインテグレーション
- 柱3：実用的な実装
- 柱4：コンプライアンスと開発の間の架け橋
- 柱5：自動化
- 柱6：測定、監視、報告、および行動

これらそれぞれの柱を支える成功したソリューションについては、クラウドセキュリティアライアンスとSAFECodeが共同で発行する、より詳細な資料でご確認ください。本文書は、上記の出版物に続くものです。

はじめに

0.1 背景

DevSecOps自動化の柱では、再帰的セキュリティを実現するためのセキュリティ自動化の利用を提唱しており、具体的には、サイバー脅威の特定、保護、検出、対応、および回復のためのセキュリティコントロールの自動化とプログラム的な実行・監視のためのフレームワークの実装です。

自動化はDevSecOpsの重要な要素であり、プロセスの効率化ならびに、開発者、インフラストラクチャーおよび情報セキュリティチームが、複雑な成果物に対して手作業やエラーを繰り返すのではなく、価値の提供に集中できるようにします。

この文書は、継続的ソフトウェア開発のデプロイメントサイクルを通じて、自動化されたセキュリティアクションを紐づける、リスクベースのセキュリティ自動化アプローチに焦点を当てています。自動化できる活動の例としては、アプリケーション、ホスト、およびコンテナの脆弱性スキャンが挙げられます。

継続的インテグレーション、継続的デリバリー、Infrastructure as Codeなどのベストプラクティスを活用するDevOpsチームは、ユーザーの要件に動的に対応し、機能を段階的にリリースし、より速いペースでデリバリーする能力を備えたアジャイルなチームと言えます。

このプロセスに情報セキュリティコントロールを統合するには、適時で意味のあるフィードバックを提供するための、自動化されたセキュリティ機能が重要です。

今日のクラウド・インフラストラクチャーは複雑であるため、小さなコードの変更が下流に不均衡な影響を与える可能性があります。したがって、セキュリティチェックは、設計、実装、テスト、リリースなど、ソフトウェアの開発と配備のライフサイクルを通じて統合され、本番環境で監視される必要があります。

再帰的セキュリティという「プラグマティック・アプローチ」の柱に沿う形で、セキュリティ機能を暫定的かつ適度に自動化することで、迅速なフィードバックが可能になり、あらゆる種類のリスクを排除できる可能性があります。例えば、OSのセキュリティ強化を確認するためのコンテナスキャンや、既知のCVE（Common Vulnerabilities and Exposures）に対するソフトウェア構成分析などです。

0.2 目的

この文書は、自動化によってセキュリティをソフトウェア開発ライフサイクルに透過的に統合するためのフレームワークを下記の点から提供します。

- セキュリティ関連情報を DevOps チームに迅速に提供し相互交流することで、セキュリティを納品への協力を妨げる1つの障害として先送りするのではなく、継続的な設計によるセキュアなコードの作成と検証を行えるようにします。
- ソフトウェア開発とセキュリティのニーズに対してバランスの取れたアプローチを可能にし、自動統合によってデリバリー効率を向上させます。

0.3 想定読者

本書の対象者は、リスク、情報セキュリティ、および情報技術の管理・運用機能に携わる人々です。これには、C-suite（CISO、CIO、CTO、CRO、COO、CEO）、特に次の機能分野に関与する個人が含まれます。DevSecOps、自動化、DevOps、品質保証、情報セキュリティ、ガバナンス、リスク管理、変更管理、およびコンプライアンス。

1. スコープ

本書では、DevSecOpsの柱である自動化について、セキュリティ自動化の必要性、セキュリティテスト自動化の手法、および、それを実現するためのメカニズムについて解説します。

この文書では、特に、DevSecOps に含まれるセキュリティ自動化を促進するための全体的なフレームワークと、セキュリティコントロールの自動化におけるベストプラクティスを提供し、DevSecOps のコンテキストにおけるセキュリティテストに関するよくある誤解を明らかにします。

2. 引用文書

次の文書は、その内容の一部または全部が、本書の要求事項を構成するような形で本文中から参照されています。日付のある文書については、引用された版のみが適用されます。日付のない文献については、参照した文書の最新版（改訂を含む）が適用されます。

- ISO/IEC 27000:2018，情報技術--セキュリティ技術--情報セキュリティマネジメントシステム--概要及び語彙
- 再帰的セキュリティによるCSA情報セキュリティマネジメント
- DevSecOpsの6つの柱

3. 用語と定義

この文書では、ISO27000、および再帰的セキュリティによるCSA情報セキュリティ管理で示される用語と定義、および以下が適用されます。

3.1 Software Composition Analysis (SCA)

アプリケーションのソースコードやコンパイルされたコードを解析し、既知の脆弱性を持つソフトウェアコンポーネントの有無を確認するセキュリティテストです。

注記1：ソフトウェア構成分析におけるソフトウェアコンポーネントには、オープンソース、ライブラリ、および共通コードが含まれる場合があります。
注記2：既知の脆弱性は、CVEなどの脆弱性データベースを介して発見される可能性があります。

3.2 Static Application Security Testing (SAST)

アプリケーションのソースコードを用いる、ソフトウェアの脆弱性やベストプラクティスに対するギャップを解析するセキュリティテストです。

注記1：静的解析は、開発者のIDE、ソースコード、およびバイナリコードなど、複数の環境で実行できます。

注記2："ホワイトボックステスト"とも呼ばれます。

3.3 Dynamic Application Security Testing (DAST)

アプリケーションの機能を実行することで実行中のアプリケーションを分析し、アプリケーションの挙動や応答から脆弱性を検出するセキュリティテストです。

注記1："ブラックボックステスト"とも呼ばれます。

3.4 Interactive Application Security Testing (IAST)

アプリケーションと一緒に配備され、アプリケーションの挙動を評価し、現実的なテストシナリオでテストされるアプリケーション内の脆弱性を検出するソフトウェアコンポーネントです。

3.5 Runtime Application Security Protection (RASP)

攻撃の検知、警告、およびブロックのために、本番のターゲットアプリケーション内に展開されるセキュリティ技術です。

注記1：WAFに似ていますが、アプリケーション内に搭載されています。

3.6 Web Application Firewall (WAF)

HTTPトラフィックを検査し、攻撃を監視、警告、およびブロックするアプリケーションファイアウォールです。

3.7 Cloud Security Posture Management (CSPM)

攻撃に対し脆弱なクラウド・インフラストラクチャーの設定ミスを発見・評価・解決できるセキュリティ技術です。

3.8 Cloud Security Monitoring and Compliance

仮想サーバを監視し、データ、アプリケーション、およびインフラストラクチャーのセキュリティリスクを評価するセキュリティ技術です。

3.9 Threat Modeling

リソースまたはリソース群に影響を及ぼす脅威を特定し、理解するための方法論です。

注記：脅威のモデル化には、STRIDE（Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, Elevation of Privilege）やOCTAVE（Operational Critical Threat, Asset, and Vulnerability Evaluation）などの手法が一般的です。

3.10 Manual Security Code Review

ソースコードを読み込み、セキュリティ上の課題を特定する人為的なプロセスです。

3.11 Software Delivery Pipeline

ソフトウェアの構想から配備までを提供する自動化されたプロセス群です。

3.12 CSA DevSecOps Software Delivery Pipeline (CDDP)

DevSecOpsの原則に沿った、セキュリティ対応が含まれるソフトウェアデリバリーパイプラインです。

4. CSA DevSecOps ソフトウェアデリバリーパイプライン

4.1. 概略

あらゆるソフトウェアは、構想、設計、開発、構築、およびテストの段階を経て、本番環境に展開されます。現代のソフトウェアデリバリーパイプラインでは、ステージごとのチェックポイントを守るために、ツールは大幅に自動化されています。パイプラインの各段階で自動的なチェックを行い、次の段階への品質課題の流出を防止します。

ソフトウェアデリバリーパイプラインには、ソフトウェア開発、変更管理、構成管理、およびサービス管理など複数のプロセスが含まれ、潜在的には、継続的インテグレーション、継続的デリバリー、および継続的モニタリングの概念が適用されます。開発・運用で使用するツールには、ソース管理、ビルド、継続的インテグレーション、コンテナ化、構成管理、オーケストレーション、およびモニタリングなどがあります。

ソフトウェア開発パイプラインにセキュリティテストを統合するには、セキュリティ活動を完全に自動化し、デリバリーパイプライン内のチームがすでに採用しているネイティブツールやプロセスとの橋渡しが必要です。チェックポイントで要求されるものの、その結果には手作業が必要な活動（本書でいう非同期テスト）は、自動化されたセキュリティチェックポイントの利点が失われないよう、特別に考慮される必要があります

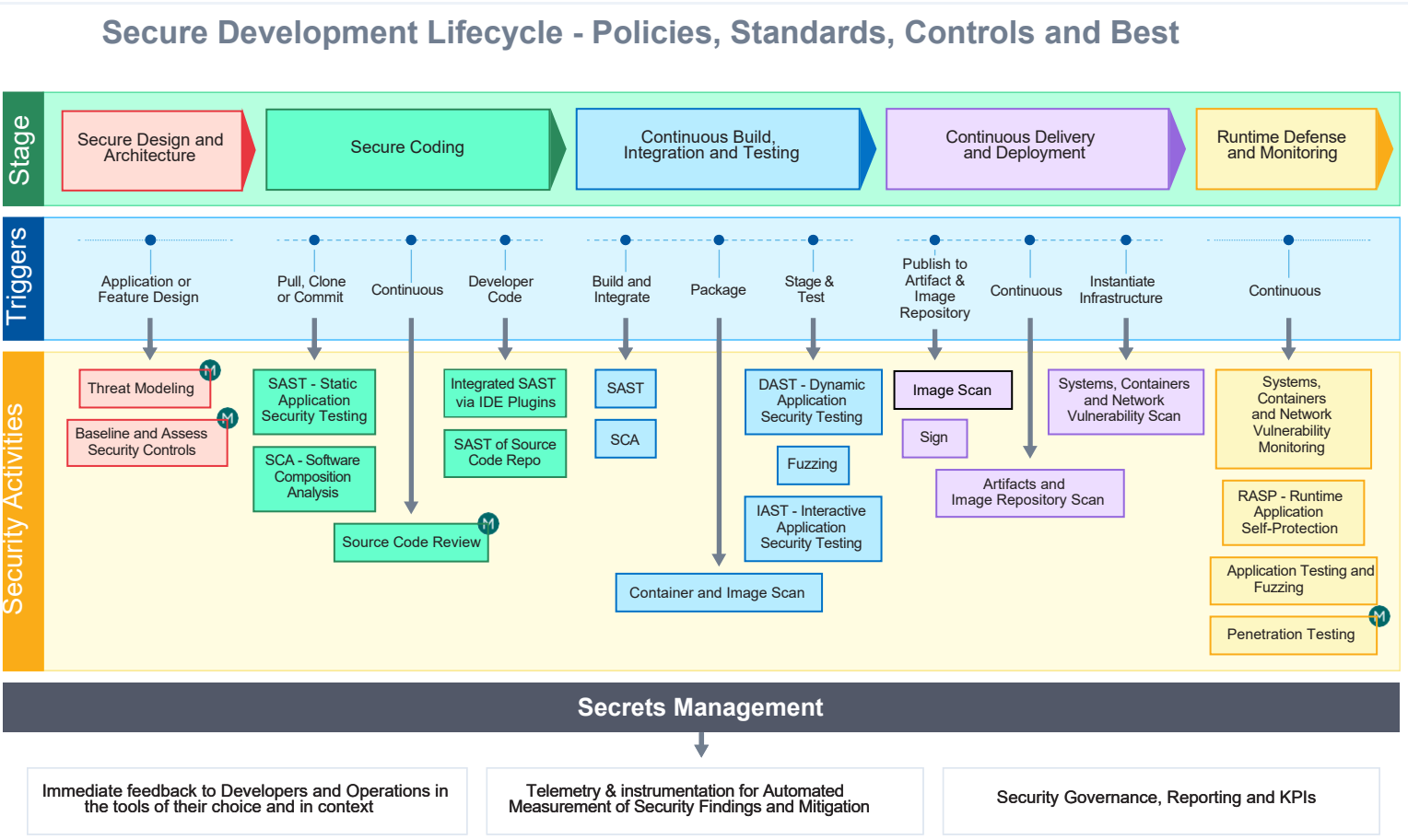

Figure 1:The CSA DevSecOps Delivery Pipeline

4.2. 構造

4.2.1. 概略

セキュリティに対応したソフトウェアデリバリーパイプラインは、3つの粒度レベルを持つと考えられています。

- ステージ
- トリガーとチェックポイント
- 活動

4.2.2. ステージ

Figure 1-1:Stages in the Delivery Pipeline

ステージは、成果物に関する段階的な成熟度向上を表します。構想時やアーキテクチャ設計時の成熟度は低いものです。ソフトウェアの継続的デリバリーとデプロイメントができるようになったとき、そのソフトウェアは明らかにより成熟しています。

あらゆるソフトウェアは、最終的に本番環境に配備される前に、設計、コーディングからテストの段階を経る必要があります。これは、ソフトウェア開発方法論（ウォーターフォール、アジャイル）から独立しています。

どの段階においても、適切なセキュリティコントロールを組み込み、自動化する機会があります。本書では、以下のようなはっきりとしたステージを特定しています。

- セキュアなデザインとアーキテクチャ
- セキュアコーディング（Developer IDEとCode Repository）
- 継続的なビルド、統合、およびテスト
- 継続的なデリバリーとデプロイメント
- 継続的なモニタリングとランタイムディフェンス

4.2.3. トリガー

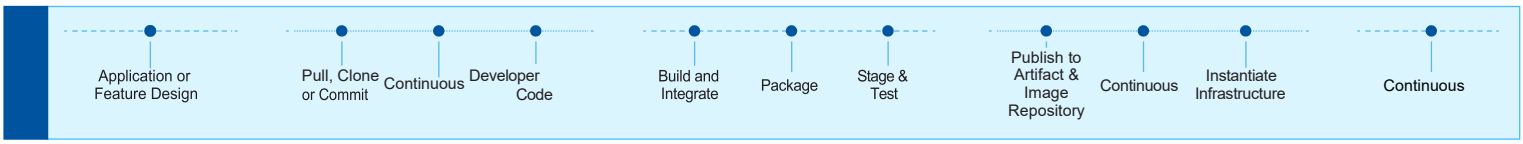

Figure 1-2:Triggers in the delivery pipeline

トリガーとチェックポイント（Figure 1-2参照）は、ステージ内の遷移を表します。トリガー条件が満

たされると、単一または複数のセキュリティ活動がアクティブになり、その活動の結果によって、チェックポイントに必要な要件が満たされているかどうかが決まります。

セキュリティ活動の結果が期待通りに要件を満たしていれば、成果物は次のチェックポイント（チェックポイントが最後の場合は次のステージ）へ移行します。そうでない場合は、成果物は依然として残り、次のチェックポイントに進むことはできません。

例1:新機能を設計することをきっかけに脅威モデリングを開始し、当該機能に適用可能な一般的または業界固有のセキュリティコントロールを特定できます。

例2:開発者によるコードのコミットやプル／マージ要求が、ソースコードのスキャンやコードレビュープロセスの自動的なトリガーとなる場合があります。

例3:コンテナイメージのコンテナレジストリへのプッシュによって、イメージがレジストリで利用可能になる前に脆弱性スキャンをトリガーする場合があります。

トリガーには2種類あります。

変更ベースのトリガー：このトリガーは、コードプッシュのような変更イベントによって生じます。
反復トリガー：このトリガーは、デイリータイマーなどの時間的なイベントによって生じます。

変更ベーストリガーは、チェックポイントの遷移を保護するために使用されます。一般的には、単一のイベント毎に実行可能な短時間のセキュリティチェックに使用されます。

反復トリガーは通常、バッチ処理やコンポーネント間の統合など、実行に比較的長時間を要するセキュリティチェックに使用されます。

例4:誤って配置されているシークレットを検索するための、複数のソースコードリポジトリに対するバッチ・スキャンには、反復トリガーの使用が推奨されます。

4.2.4. 活動

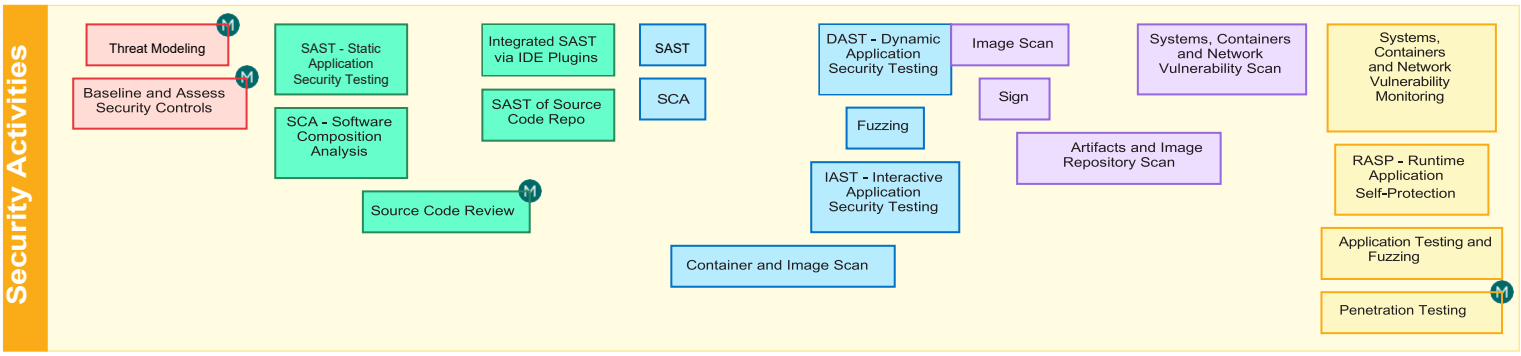

Figure 1-3:Security Activities in the delivery pipeline

活動とは、成果物を入力とし、ポジティブまたはネガティブな結果を出す、個々のプロセスのことです。成果物が活動の基準を満たす場合、その成果はポジティブであると考えられます。そうでない場合は、結果はネガティブになります。

具体的には、各活動は、セキュリティコントロールの実装を表しており、コード、ライブラリ、アプリケーションなどに含まれる、あるいはアプリケーションを実行する環境におけるセキュリティ脆弱性を検出するために実行されるテストまたは検証タスクです。

活動はステージやトリガーから独立していますが、一部の活動はいづれのステージで実行できる場合もあれば、1つのステージに拘束される場合もあります。複数のトリガーは、同一な活動の結果に依存する場合があります。各セキュリティ活動の詳細な説明とガイドラインは、参照ドキュメントをご覧ください。

4.3. パイプラインの設定と保守

DevSecOpsをサポートするために、デリバリーパイプラインは、DevOpsチームがすでに使用しているネイティブツールやプロセスに組み込まれる必要があります。摩擦を最小限に抑え、効率的で迅速なリリースサイクルをサポートするために、セキュリティは通常のリリースプロセスの一部である必要があります。

現実的なセキュリティの観点からは、パイプラインのセキュリティテスト機能は、価値が高く影響が少ない機能を最初に追加し、段階的に導入される必要があります。すでに使用されているコード品質やライブラリ管理[1]ツールからセキュリティチェックを行うことで、運用への影響や追加リソース（ベンダーの特定、新しい技術の取得、および組み込み）を最小限に抑えながら、素早く効果を得られます。

新しいテスト機能、たとえばStatic Application Security Testing (SAST)を導入するときに、DevOpsチームは、一例として、まず最初に特定の発見タイプに焦点を当て、誤検知を防ぎ、最大のコードカバレッジ（偽陰性の排除）を確保するように、ツールを調整する必要があります。このチームは、新しい検出タイプや機能を追加する前に、見逃していた脆弱性や誤検知の根本原因分析を行い、自動テストの効率を最適化できます。

大量の誤検知はDevOpsチームの負担となり、セキュリティツールの信頼性を損ないます。

良い戦略とは、「セキュリティを左へシフトしつつ、右へ加速する。」ことです。 自動化されたセキュリティコントロールと活動は、組織の既存のSDLCポリシー、標準および手続きを置き換えるものではありません。むしろ、自動化はプロセスや資産を保護するための、既存の能力を補強するものです。組織は、手動で行われているコントロールと活動を自動化することから始める必要があります。

5. リスク観点から優先順位付けられたパイプライン

5.1. 概略

すべてのアプリケーションが同一のリスクプロファイルを持つものではありません。リソースの優先順位を効率的に決め、デリバリー量とのバランスを取るためには、安定した履歴のあるアプリケーショ

[1] https://www.csoonline.com/article/3398485/28-devsecops-tools-for-baking-security-in- to-the-development-process.html

ンのデリバリーを迅速に行う一方で、リスクの高いアプリケーションのデリバリーをより精査する仕組みを導入することが有益であると考えられます。

ビルドの配備時に実施するセキュリティテストの全体的な厳格さを決定するために、リスクベースのアプローチを使用する必要があります。リスク要因には、一般的に次の3種類があります。

5.2.リスク要因

5.2.1. アプリケーションのリスク

技術的な露出に関係なく、アプリケーションが処理するデータとその使われ方の背景を含む、アプリケーションに起因する固有のリスクです。

例:銀行やヘルスケアデータを扱うアプリケーションは「高リスク」と考えられ、個人情報やクレジットカードを扱うアプリケーションは「中リスク」と考えられます。 静的なコンテンツや公開情報を扱うアプリケーションは、「低リスク 」と考えられます。

5.2.2. 変更案のリスク

変更による潜在的なセキュリティへの影響、変更によって影響を受けるデータ、および変更の性質を考慮した、変更自体に起因するリスクです。

例:認証やアクセス管理のような中核的なセキュリティコンポーネントへ影響を与える変更、新しい機能や技術を導入する変更、または再構築を伴う変更は、アプリケーションのセキュリティ態勢に影響を与える可能性が高いため、より徹底的にテストされる必要があります。

5.2.3パイプラインとその成果物の信頼性履歴からのリスク

デリバリーパイプラインおよび当該アプリケーションの信頼性および完全性の履歴が要因となるリスクです。

例:セキュアな納品物の歴史を持つ安定したデリバリーパイプラインは、脆弱な納品物の歴史を持つ不安定なデリバリーパイプラインよりもリスクが低いことは明らかです。安定したセキュアなリリースが確認できているアプリケーションは、セキュリティ欠陥が相次ぐアプリケーションよりもリスクが低いのです。

5.3パイプライン構成の優先順位付け

デリバリーパイプラインの構成は、リスク要因の総体にもとづいて優先順位付けできます。デリバリーパイプラインやアプリケーションはそれぞれ異なるため、実装にあたっては、リスク鑑別の仕組みがユースケースに対し適切で十分であるかどうかを検討する必要があります。

デリバリーパイプラインに対する区別された扱いが存在することで、セキュアに開発・展開されたアプリケーションに対して等級分けされたインセンティブを与え、デリバリー時間の短縮につなげることができます。

本節は、信号機メタファーを活用して、グリーン、イエロー、レッドの3色の処理レーンを設け、それぞれ異なるレベルのセキュリティ精査を行います。

- グリーンレーン：継続的デリバリーに最適化された短い実行テスト
- イエローレーン：DASTを含む可能性があり、処理時間の短いマイナーな帯域外の活動、継続的デリバリーが可能なもの。
- レッドレーン：IASTを含む可能性があり、処理時間が長いメジャーな耐域外の活動、継続的デリバリーが困難なもの。

例1:3つのリスク要因のすべてで低ランクのアプリケーションは、「全体的に低リスク」と考えられ、「グリーンレーン」に沿って継続できます。

例2:低リスクのアプリケーションでの、個人データの共有に影響を与えない個人データ取扱コードの変更は、「中リスク」と考えられ、「イエローレーン」に沿って継続できます。

例3:重大な脆弱性の記録を持つアプリケーションのデリバリーは、「レッドレーン 」を用いて、より厳格なセキュリティの対象とする必要があります。

例4:提案された変更が、データの安全な取り扱いのための新しい標準コンポーネントを導入する場合、または開発者に対するセキュリティのベストプラクティスに関するトレーニングを要する場合は、「レッドレーン」が適用されます。

新たな脆弱性が発見された場合、その原因が人、プロセスまたは技術にあるのか、再発防止のために何を再設計・変更すれば良いのか、根本原因を分析する必要があります。

下の表は、これら3つの要素を組み合わせて、目標とするパイプライン構成を決定するための変更のスコアリングの例です。表中では、高（3）、中（2）、および低（1）の数値が用いられ、合計点が5点以上を「イエローレーン」、および7点以上を「レッドレーン」と考えています。

	Application Risk	Change Risk	Reliability Risk	Delivery Pipeline Configuration
Change #1	Medium (2)	Low (1)	Low (1)	Green (4)
Change #2	Low (1)	High (3)	Medium (2)	Yellow (6)
Change #3	High (3)	High (3)	High (3)	Red (9)

Table 1:Build Impact and Resulting Release Pipeline

6. デリバリーパイプライン活動フレームワーク

6.1. 概略

本セクションでは、デリバリーパイプラインで使用されるセキュリティ活動の5つのクラスを表すフレームワーク（FIgure 2を参照）と、それらの各クラス内の活動に関するガイドラインを示します。

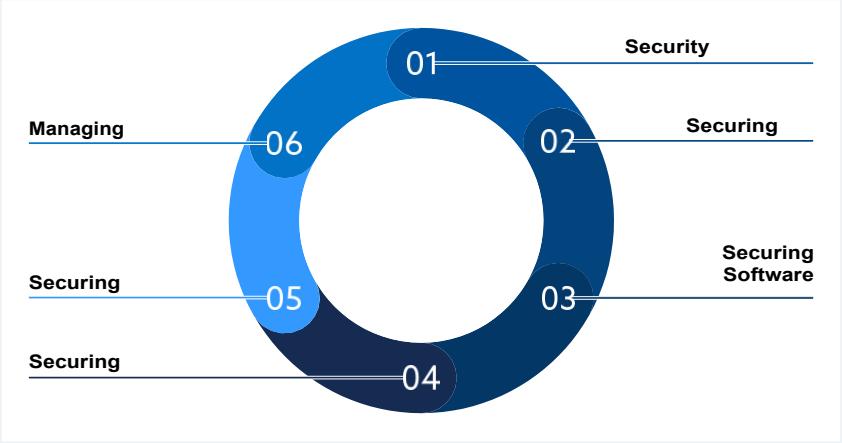

Figure 2:Security Activities in the Security-enabled Delivery Pipelines

6.2. 設計のセキュア化

セキュリティは、設計されて初めて、実現されるものです。後から思い付きで適用するセキュリティ対策は災いの元になります。設計上の欠陥が、脆弱性の大部分を占めていることが知られています[1]。

手動プロセスはDevSecOpsの実践と相容れないと考える人もいますが、そうではありません。セキュア・アーキテクチャの確立を確保するために、脅威のモデル化などの人間の知性を必要とする、中核的なセキュリティ設計活動があります。

従来は脅威のモデル化は手動でしか行えませんでしたが、今日ではこのプロセスを自動化するためのツールがあります。さらに脅威モデルは、アプリケーションに新しいコンポーネントが追加されたり、セキュリティ上の重要なコンポーネントが変更されたりしない限り、頻繁な更新を通常では必要としないため、手動更新は依然として有効なメカニズムです。

6.3. コードのセキュア化

脆弱性は、アプリケーションのセキュリティアーキテクチャが熟考され、かつ、堅牢であったとしても、不適切なコーディングから発生する可能性があります。開発者が書いたコードにセキュリティ上の欠陥があれば、アプリケーションが侵害されるのは時間の問題です。

SASTツールは、成果物のソースコードをスキャンし、セキュリティ（または一般的な）欠陥を報告し、それらがセキュリティ脆弱性として積込まれることを防止するために役立ちます。

静的コードテストを自動化する際のDevSecOpsの目標は、コードセキュリティアラートをできるだけ早く、文脈に即し、かつ規範的に開発者へ提供し、開発者へのフィードバックループを強化することです。

デリバリーパイプラインには、下記のようなセキュリティスキャンを行う機会が数多く存在します。

- ローカルマシン上で：IDEまたは統合テストスイートを介して
- コードスキャンを伴う継続的インテグレーション：継続的インテグレーションシステムでのコードプッシュ、マージ、およびリリース時にコードスキャンを実行
- 継続的デリバリーのプリ・デプロイ：デプロイ前のスキャン作業

6.4. ソフトウェアコンポーネントのセキュア化

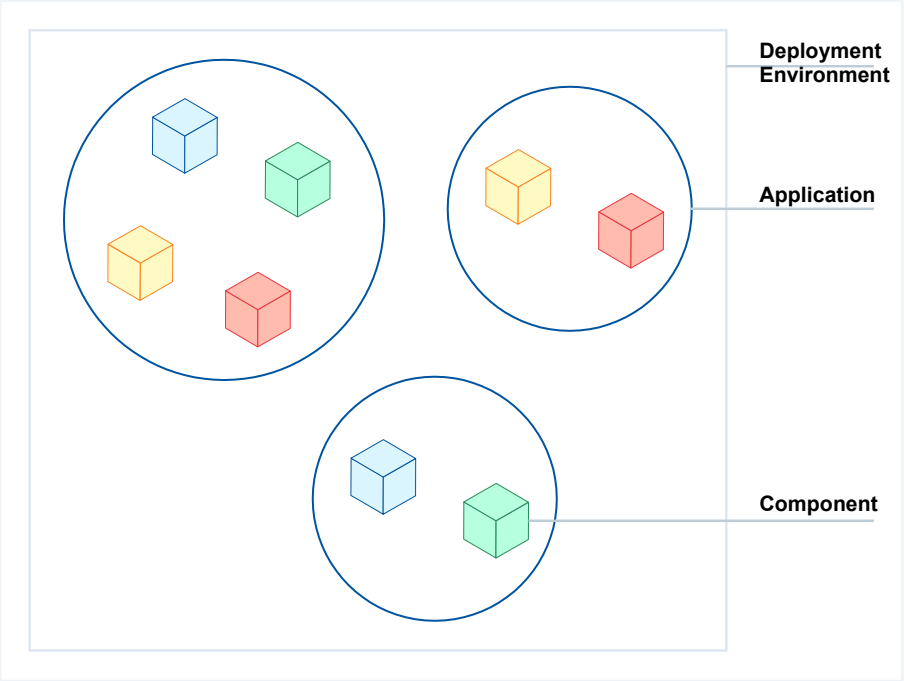

Figure 3:Relationship among Application, Application Components and Environment

アプリケーションは、その構成要素のセキュア性によってのみセキュアになります。そのコンポーネントの脆弱性は、自社製、サードパーティ製、またはオープンソースプロジェクト製にかかわらず、大きなアプリケーションへ侵害をもたらす可能性があります。

下記のような対策を講じ、これらのソフトウェアコンポーネントを体系的に管理することが重要です。

- オープンソースやコンテナなどの、サードパーティソフトウェア内の脆弱性を迅速に検出・報告する
- 未承認なコンポーネントの使用を検出する

これらのコンポーネントの脆弱性管理は、開発期間中だけでなく、その後も継続的に行う必要があります。使用するサードパーティコンポーネントは、アプリケーションのライフタイムを通じてセキュア

であることの確保が重要です。

アプリケーションのコンポーネントの脆弱性が公開された場合、コンポーネントライブラリを脆弱性の無いバージョンにアップグレードする必要があることを示すプロセスがトリガーされる必要があります。

このプロセスは、継続的インテグレーションプロセスやランタイムスキャンを通じて、ソースコードスキャンツールを用いて自動化できます。

6.5. アプリケーションのセキュア化

アプリケーションが信頼されるための前段階として、現実的な環境でのセキュリティテスト実施が必要です。強固なアーキテクチャ、セキュアなコード、および脆弱性のないコンポーネントがあっても、それらを統合することだけで防弾されたアプリケーションになるとは限りません。

ダイナミックアプリケーションテストは、確信を得るために、ステージング、テスト、および本番環境でのアプリケーションのテストに使用できます。DASTツールは、障害注入、障害検出、リソースやシークレットの漏洩など、様々なセキュリティ課題を発見できます。(パイプライン内におけるDASTの適切な配置については、Figure 1を参照してください)。

また、ファジングはセキュリティ脆弱性を発見するための有効な手法です。誤った入力や悪意のある入力に対するアプリケーションのレジリエンスを確保するために、デリバリーパイプライン内で実行可能なさまざまな種類の自動化ファジングが利用可能です。

6.6. 環境のセキュア化

概略

敵対的な環境は、セキュアなアプリケーションを非セキュアな状態にします。Infrastructure as Code(IaaC)、コンテナ化されたアプリケーション、および継続的デプロイのアプローチは、インフラストラクチャーや環境のセキュア化に関し、次のような新しい可能性をもたらします。

- IaaCのコードと中間生成物のスキャニング
- 環境内の実行時ディフェンス

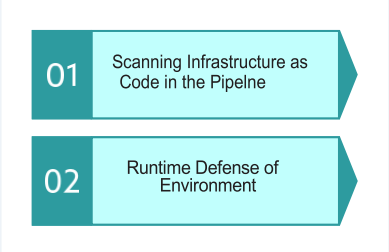

Figure 4:Multi-pronged Approach to Securing Infrastructure

IaaCセキュリティ

IaaCのコードは、クラウド・インフラストラクチャーのリソースの宣言型管理に用いられ、IaaCの構成状態は、当該リソースの実際の構成に関する正確なリフレクションとすることができます。IaaCのコードと中間生成物のセキュリティ解析は、2つのメカニズムで行うことができます。

- IaaCテンプレートと構成状態の静的スキャン
- 宣言的にデプロイされたリソースの自動スキャン

ランタイムセキュリティ

デプロイされたアプリケーションは、その環境のセキュリティに左右されるため、監視が必要です。このためには、一般的に2つのアプローチがあります。

- デプロイされたアプリケーションと環境のランタイムモニタリングは、当該の環境とアプリケーションの両者の保護にするための重要な対策です（Figure 4参照）。
- ランタイムディフェンスは、実行時に発見された脆弱性や攻撃を可及的速やかに緩和され、報告されることを確実にします。

6.7. シークレットの管理

アプリケーションとデリバリーパイプラインにおけるシークレットの管理は最も重要であり、デリバリーパイプライン全体に影響を及ぼします。適切な暗号アルゴリズムの使用は、必ずしも敵対者に対する十分なセキュリティを提供するものではありません。例えば、開発者が最適な暗号化アルゴリズムを選択しても、鍵がオープンな場所に保管されていれば、セキュリティは損なわれます。

鍵の管理、鍵のローテーション等のプロセスが適切に実施されている必要があります。クラウド鍵保管庫や鍵管理インフラストラクチャー、自動化ソリューションの一部として検討できます。

7. 自動化のベストプラクティス

7.1. 概略

セキュリティ対応デリバリーパイプラインで使用される多くのセキュリティ活動は、どのDevSecOpsプロセスでも独立して使用できます。成熟した組織の多くは、アウトオブバンド（または非同期）テストの規定を持っています。このセクションでは、DevSecOps 以外にも適用可能なベストプラクティスや考慮すべき点を紹介します。

7.2. 脆弱性の緩和

脆弱性は、作られたばかりの時が最も対処が容易です。この点を考慮して、この戦略は、新たに作られる脆弱性の検知を、デプロイに入る前の開発プロセス内へ移行するものです。例えば、IDE内やコード変更をコミットする前にセキュリティ上の課題をチェックするツールなどが検討できます。

後の工程で検知された脆弱なコードについては、本番環境にデプロイされる前に当該の脆弱性を緩和することが重要です。アセスメントされた脆弱性のリスクに基づき、重要度の低い事項は後で対処しても構わないものの、重要度の高いものはできるだけ早く対処しなければなりません。

7.3. 非同期テスト（帯域外テスト）

非同期テストには、一般的に2つのカテゴリがあります。

- ある種の重要なセキュリティ活動は、人間の知性を必要とするため、完全な自動化はできません。これには、脅威のモデル化、ペネトレーションテスト、およびピアコードレビュー等が

含まれます。
- SASTのような、実行に時間がかかるヘビー級の自動テスト。

これらの活動は、ソフトウェアデプロイメントのパイプラインを中断させないように、自動デプロイメントの内部ではなく、「アウトオブバンド」でトリガーおよび実行されるようにできます。

アウトオブバンドで特定された課題は、デリバリーチームが追跡し、可視化、優先順位付け、および特定されたリスクに応じたロールバックの可能性を確保される必要があります。帯域外のテストから生じる重大な課題に時間内に対処できない場合、適切なチェックポイントにおいて継続的デリバリープロセスに対する閉塞物として計数される必要があります。

7.4. 継続的フィードバックループ

アプリケーション内にセキュリティ上の欠陥が存在している期間が長くなるほど、さらなる変更やデプロイによって、その欠陥の修正に要する複雑さとコストが増える可能性が高くなります。

DevSecOpsチームは、欠陥の下流への移行を防ぐために、セキュリティ課題に関し、できるだけ早く警告を受ける必要があります。

さらに、タイムリーなフィードバックにより、各チームメンバーが脆弱性の原因となった変更を迅速に診断し修正できるようにし、付随するコンテキストに関する各メンバーの記憶がまだ新しいうちに根本原因の分析を実行できます。

7.5. ビルドの中断

組織は、自動化されたデプロイメントパイプラインを中断することとなりますが、セキュリティ要件を満たさない場合の「壊れたビルド」を宣言できるようにするために、セキュリティ活動がビジネス要件であることを認識する必要があります。

セキュリティのためにビルドを中断することは、品質を強制し、DevSecOpsチームに迅速なフィードバックを提供し、組織のリスク許容範囲を超えるデータ露出や潜在的なエクスプロイトを防止します。例えば、重大性が高く、エクスプロイトが活発な公知の脆弱性を持つコードをリリースする場合などです。

ビルドを中断させる可能性がある活動は以下を含みます。

- 組織のリスク許容度を超える重大な欠陥が確認された場合。
- 組織のポリシーに抵触する特定の脆弱性または脆弱性クラスが特定された場合。
- 高いもしくは重大な発見事項の数が一定の閾値を超えた場合。
- 必要なセキュリティ活動が効果的に実行されない場合。例えば、設定エラーにより自動スキャンが失敗している場合。

8. 結論

DevSecOpsと再帰的セキュリティのアプローチの恩恵を受けるには、自動化に大きく依存する必要があります。

自動化の柱は、セキュリティに対応したデリバリーパイプラインと、その中でのセキュリティコントロールの

適用によって実現できます。また、このリスク優先のアプローチにより、継続的デリバリーへの最適化や、自動化されていないセキュリティチェックに対応する仕組みを可能にします。

参考文献

1. 3 security risks that architecture analysis can resolve.Synopsys, 2016.https://www. synopsys.com/blogs/software-security/security-risks-that-architecture-analysis-can-resolve/から入手できます。

2. Tactical Threat Modeling.SAFECode, 2017. https://safecode.org/wp-content/uploads/2017/05/SAFECode_TM_Whitepaper.pdf から入手できます。

3. Managing Security Risks Inherent in the Use of Third-party Components.SAFECode, 2017. https://safecode.org/wp-content/uploads/2017/05/SAFECode_TPC_Whitepaper.pdf から入手できます。

4. Focus on Fuzzing.SAFECode, 2020. https://safecode.org/category/focus-on-fuzzing/ から入手できます。