B-022

DevSecOps におけるパイプラインのセキュリティ検証手法
Pipeline security verification method in DevSecOps

吉村 隼哉† 桑名 栄二†
Shunya Yoshimura Eiji Kuwana

1. はじめに

近年,ソフトウェアサプライチェーン（SSC）を狙ったサイバー攻撃が増加しており，企業の開発・運用環境を狙った攻撃も確認されている．こうした状況の中，クラウドネイティブな IT システムの普及に伴い，柔軟なシステム開発を実現する DevSecOps[1]の導入が求められている．

DevSecOps においては，業務の効率化や自動化のために，多くのクラウドサービスやオープンソースソフトウェア（OSS）が導入され，DevSecOps 環境自体が 1 種の複雑な IT システムを形成している（図 1）．そのため，開発アーティファクトだけでなく，DevSecOps 環境自体にもセキュリティ対策が必要であることが Woody ら[2]によって示されている．

しかしながら，DevSecOps 環境（パイプライン）全体を対象とした包括的なセキュリティフレームワークは依然として確立されておらず，開発組織は複数の断片的な対策を組み合わせて導入せざるを得ない現状が Okafor らの研究[3]により明らかにされている．

SSC の管理においては，経済産業省より「ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引」[4]が発出されるなど，SBOM の導入が推進されている．しかし，頻繁なアップデートや自動更新が行われるクラウドサービスに対する SBOM への反映には課題があることが，CISA（Cybersecurity and Infrastructure Security Agency）が発行したファクトシート[5]によって示されている．DevSecOps パイプラインは多くのクラウドサービスから構成されるため，このような環境への SBOM 適用にも課題が残る．

こうした背景から，本研究では DevSecOps パイプライン全体を対象とした新たな SSC 検証手法の提案を目指す．具体的には，パイプラインの SSC を可視化する PBOM（Pipeline Bill of Materials）の概念[6]を取り入れたパイプライン管理と，来歴による SSC 検証を実現するフレームワーク（in-toto[7]）を活用した DevSecOps パイプライン全体の SSC 検証手法を提案する．

PBOM とは，SBOM の概念を拡張し，パイプラインを対象とするソフトウェアコンポーネント管理を実現するリストのことを指すが，現時点において具体的な要素や運用方法については標準化されていないのが現状である．

本稿では，代表的な SSC 攻撃の事例と既存の関連ガイドラインおよび関連フレームワークについて述べ，既存手法の課題を明らかにした上で，PBOM を活用したパイプラインの SSC 検証手法を提案する．さらに，提案手法に基づく具体的な実現例を示し，今後の展望について述べる．

† 情報セキュリティ大学院大学 Institute of Information Security

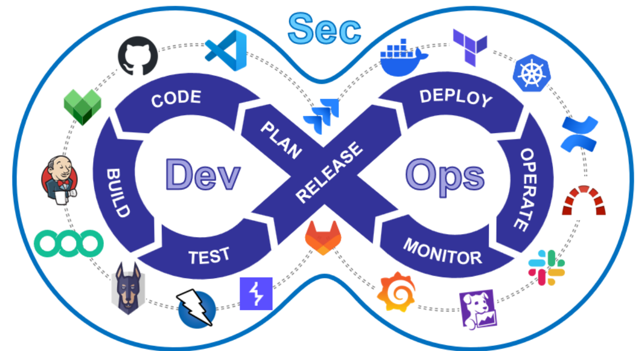

図 1 DevSecOps とツールの全体像

2. 研究の背景

DevSecOps において，SSC のセキュリティ検証が求められるようになった背景として，DevSecOps パイプラインが複雑化した要因と，パイプラインの SSC リスクに関連した 3 つの代表的なセキュリティインシデント事例を以下に示す．

2.1 DevSecOps の普及とパイプラインの複雑化

クラウドネイティブなシステムの普及に伴い，現代のシステム開発では，スピードと柔軟さが求められるようになっている．Linux Foundation が実施した調査[8]によると， 1 日に複数回のコードリリースを実施している組織が 29%に達し，毎日，毎週のリリース頻度を実施している企業を合わせると，60%を超えていることが示されている．

このような高頻度のリリースは，従来のウォーターフォール型開発手法（リリース頻度が数か月程度）では実現が困難であることから，アジャイル開発の採用が進みつつある．さらに，開発・運用業務の効率化・自動化を目指す DevOps，セキュリティの要素も統合する DevSecOps が登場している[9].

DevSecOps の起源は，Gartner 社が DevOps にセキュリティが欠かせないとし，DevOpsSec として 2012 年に提唱したことがはじまりとされている[10].

DevSecOps が求められるようになったことで，企業は DevSecOps プラットフォーム（パイプライン）への投資を重要視するようになり，GitLab が実施した調査[11]によると，企業における投資先の優先度として DevSecOps プラットフォームが上位に挙がっていることからも，DevSecOps への期待が表れている．

また，Zhao らの研究では[12]，DevSecOps の課題として Technology の分野に多くの課題があることを示しており，その範囲はソフトウェア開発ライフサイクルの全体に跨るとされている．つまり，DevSecOps パイプラインの構築においては，ツールの適切な運用の観点で多くの課題があることが示されている．

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

2.2 ソフトウェアサプライチェーン（SSC）の脅威

IPA の「情報セキュリティ 10 大脅威 2025」[13]によると，サプライチェーンや委託先を狙った攻撃は，組織を対象とする脅威の中でランサム攻撃による被害に続く 2 位に位置づけられている．ここで，SSC の定義は「ソフトウェア開発ライフサイクルに関与する全てのモノ（ライブラリ,各種ツール等）や人の繋がり」とされており，主な攻撃手口として，ソフトウェア開発元や MSP（マネージドサービスプロバイダー）等を攻撃し,標的組織を攻撃するための足掛かりとする手口が紹介されている．

SSC に対する攻撃の対策としては，①信頼できるサービス選定，②納品物の検証などが挙げられている．①においては，ISMAP などのサービス信頼性評価により，複数候補からの選定が求められている．②においては，納品物に組み込まれているソフトウェアの把握と脆弱性対策が求められ，SBOM の導入が推奨されている．

本稿では，SSC 攻撃の代表的な事例である Solarwinds 社の Orion，Codecov，Apache log4j2 の事例を元に脅威を示す．

2.2.1 SSC 攻撃事例①（Orion）

Solarwinds 社が提供するネットワーク管理ソフトである Orion において，開発環境が侵害されたことで発生した事案である．

攻撃者は，サードパーティ製品のゼロデイ脆弱性攻撃，ブルートフォース攻撃，ソーシャルエンジニアリングなどの攻撃手法を用いて Orion の開発環境に侵入し，正規のアップデートプログラムにバックドアを設置した．そして，Orion のアップデート後にバックドアを介して Orion ユーザーのネットワーク侵入に成功している．米国連邦政府も被害にあうなど，SSC のセキュリティリスク（SSC 汚染）の観点から大きな影響を及ぼした事案である[14]．

この事案から，開発環境への侵入が SSC 攻撃の入口となることが示され，開発環境の保護が重要であることが再認識された．

2.2.2 SSC 攻撃事例②（Codecov）

ソースコードのテストカバレッジ可視化ツールである Codecov の脆弱性が悪用された事案である．

攻撃者は，Codecov の Docker イメージ作成方法の脆弱性から認証情報を入手し，開発環境へ侵入することで，Codecov ユーザーが使用するアップロードスクリプトに不正なコードを挿入した．不正コードが含まれる Codecov を利用したユーザーから情報搾取に成功している [15]．

攻撃の入口は Orion（2.2.1）の事例と同様，開発環境に侵入され，不正コードを挿入されたことが原因である．さらに，Codecov は開発ツールとして利用されていたことから，Codecov を導入するシステム開発企業が被害にあったことが特徴である．

この事例から，開発環境で利用するツールの管理と検証が重要であることが認識された．

2.2.3 SSC 攻撃事例③（log4j2）

Java のログ処理ライブラリである Apache Log4j2 の脆弱性（CVE-2021-44228）により，Log4j2 を利用するサービスに対し，攻撃者はリモートから任意のコードを実行することが可能であった．

この事例が大きく取り上げられた理由として，3 つの要因が挙げられている：

- 脆弱性スコアが最大だった
- 広く利用されている OSS だった
- 脆弱性の悪用が容易だった

また，この事例の対応に苦労した点として，SSC 汚染の範囲，つまり「影響範囲を自社で調べきれない」点が挙げられており，SBOM による外部製品の管理が重要であることが認識された事案である[16]．このことから，Codecov のようにパイプライン上のツールで脆弱性が発生した場合にも，外部製品が管理されていなければ影響範囲の特定に時間を要することが想定される．

3. 関連ガイドライン

SSC 攻撃の増加を受けて，米国大統領令を契機に，SSC セキュリティ強化に関する議論が活発化し， 様々なガイドラインが発行されている．

3.1 米国大統領令 14028（EO 14028）と NIST 文書

2 章で示したように，SSC の脆弱性を悪用したサイバー攻撃の増加・深刻化により，米国大統領令 14028 [17]では，SSC セキュリティ強化の項目において NIST に対してガイドラインの策定を命じた．

EO 14028 を受けて，作成された NIST 文書が NIST SP 800-218（Secure Software Development Framework (SSDF)）[18]である．SSDF は，セキュア開発の原則と要求事項を定義し，ソフトウェア開発ライフサイクル（SDLC）全体でのセキュリティ統合を促し，安全なコード作成・レビュー・ビルド・リリース・維持に至る各工程へのセキュリティ統合を促している．

さらに，DevSecOps においては，SSDF を満たすための戦略を示す文書として，NIST800-204D（「Strategies for the Integration of Software Supply Chain Security into DevSecOps CI/CD Pipelines）[19]が策定されている．本文書では，SBOM や SLSA（Supply-chain Levels for Software Artifacts）といった具体的な例を挙げ，SSC の対応策を示している．

3.2 ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引

米国での動きを受けて，日本国内においても経済産業省よりソフトウェア管理に向けた「ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引」[4]が発出され，SBOM の導入が推進されている．

本手引では，SBOM の作成および運用方法について提案がなされているが，対象は主に開発アーティファクトに限定されており，開発・運用環境上のソフトウェア管理については明記されていない．

また，SBOM 生成ツールに関しては，主要なプログラミング言語への対応が明示されているが，CI/CD（Continuous Integration/Continuous Delivery(Deployment)）パイプラインの記述に使用される YAML(YAML Ain't Markup Language) ファイルへの対応が明記されておらず， SBOM 生成ツール体系的に分析した Mirakhorli らの研究[20]においても YAML が対象外となっている．このことから，主要な SBOM 生成ツールでは，パイプラインを直接解析対象とする機能が備わっていない現状が読み取れる．

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

3.3 CIS Software Supply Chain Security Guide

CIS Software Supply Chain Security Guide[21]は，NIST のフレームワークやゼロトラストの原則を活用した，幅広いセキュリティ対策に関するガイドラインである．
本ガイドラインは，5 つのカテゴリに分かれ，100 項目を超える推奨事項が掲示されている点が特徴であるが，各項目を満たすための具体的フレームワークは記載されていないため，具体的な実装ガイドラインと組み合わせて対応を進める必要がある．

3.4 SLSA（Supply-chain Levels for Software Artifacts）

SLSA[22]とは，ソフトウェアアーティファクトのサプライチェーンセキュリティを段階的に確保するためのフレームワークである． SLSA は，SBOM の運用に適用することで，ソフトウェアの信頼性と整合性をより高めることが期待されている．
図 2 は，SLSA の全体像および対象となる SSC の脅威を示しており，例として Orion の事例（2.2.1）は図中 E，Codecov の事例（2.2.2）は図中 F の脅威に該当するとされている．そして，これらの脅威に対して SLSA の高レベルモデルを構築することが推奨されている．
CIS ガイド（3.3）と比較すると，より具体的な実装手順が段階的に記述されているため，実装支援の観点で有用性が高い．しかし，SLSA はビルドプロセスに焦点が当てられているため， DevSecOps パイプライン全体をカバーするためには，SLSA の対象範囲外のプロセスについて考慮が必要である．

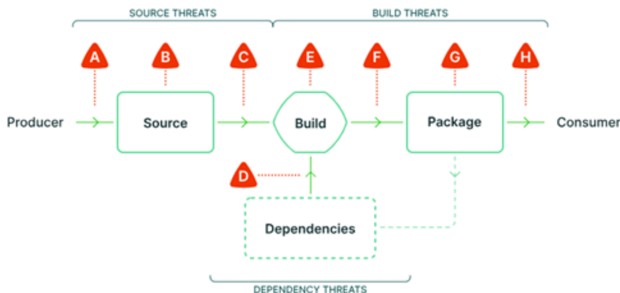

図 2　SLSA の全体像と脅威ポイント[22]

3.5 Software Supply Chain Best Practices

Software Supply Chain Best Practices[23] は，Cloud Native Computing Foundation（CNCF)から提案された，SSC 保護のためのベストプラクティス集である．
本文書では，SSC に対するリスクを軽減するための具体的な実装フレームワークが紹介されている点で，実装支援の観点から有用性が高い．
パイプラインの観点では，BuldPipeline の章において，パイプラインのコード化（パイプラインの Infrastructure as Code 化），ツールのバージョン固定，パイプラインにおける来歴情報（provenance）の生成と検証についての記載がある．また，これらの実現手段として，in-toto[7]や sigstore[24]などの具体的フレームワークが挙げられている．
なお，本文書は 2024 年 11 月に v2 が公開され，これまで紹介した SSC 関連ガイドラインの中で，最も新しい文書である．そのため，近年のセキュリティ動向や実装フレームワークを反映した，最新の SSC 対策指針となっている．
本研究では，本文書で示されたベストプラクティスの推奨事項を満たす形で DevSecOps パイプライン全体を対象とする検証手法の提案を目指す．

4. 関連フレームワーク

4.1 フレームワークの分類と比較（Okafor らの研究）

Okafor らの研究[3]によると，「SSC セキュリティの分野は依然として発展途上にあり，現時点では断片的な対策やメカニズムが多く，開発者がそれらを体系的に理解・設計に反映できるような包括的フレームワークは存在しない．」と指摘している．
同研究では，既存の SSC セキュリティ関連フレームワークを， Transparency（透明性）， Validity（正当性）， Separation（分離性）の 3 つの特性に分類し，さらに Artifacts（アーティファクト）， Operations（運用・オペレーション）， Actors（関与者・アクター）の 3 つの視点でそれぞれのフレームワークが，どの特性を満たしているかを可視化している（表 1）．
可視化の結果，既存フレームワークは Artifacts の視点に集中し，Operations，Actors に焦点を当てたフレームワークは少ないことが示されている．
本研究では，これら 3 つの特性を幅広くカバーしているフレームワーク（in-toto[7]）に注目し，DevSecOps パイプライン全体における SSC 整合性検証の実現を目指す．また，SBOM の概念を拡張し，パイプラインの SSC 管理に適用した PBOM（Pipeline Bill of Materials）の生成，およびパイプライン検証への活用を目指す．

4.2 in-toto

in-toto[7]は，SSC における各プロセスの整合性を保証するための来歴検証フレームワークである．in-toto では，パイプライン内の各ステップについて，実行されるコマンド，実行ユーザー，使用・生成されるアーティファクトなどを，Layout ファイル（JSON 形式）として定義する．
パイプラインプロセスの実行時には，Layout ファイルに定義したステップ毎に来歴情報（link ファイル）を生成することで，事前に定義した Layout ファイルを元に，定義通りのプロセスを経ているかを検証する．
in-toto は，コマンドラインインタフェース（CLI）および Python API として提供されており，本研究では将来的な拡張性を考慮し，本稿執筆時点で最新バージョンである in-toto 3.0.0 の Python API を採用する．
in-toto が提供する主な機能は以下の 3 点であり，これらの機能は SLSA の高レベルモデルの実装を支援する要素とされている：

- Layout 作成機能
- 来歴作成機能
- 来歴検証機能

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

表 1　Okafor らによって示された既存フレームワークの分類　[3]

Techniques	Transparency			Validity			Separation		
	Artifacts	Operations	Actors	Artifacts	Operations	Actors	Artifacts	Operations	Actors
SBOM	✓	✓							
npm-audit [55]	✓			✓					
Code scanning [1]	✓			✓					
Dependabot features [29]	✓			✓					
GitHub Actions [28]		✓		✓	✓			✓	
Git Commit Signing [27]			✓	✓					
Scope [54]				✓			✓		✓
Multi-Factor Authentication						✓			
In-toto [73]	✓	✓		✓	✓			✓	✓
Containerization							✓	✓	✓
Version Locking							✓		
Sigstore [51]	✓	✓	✓	✓	✓				
Mirroring and Proxies [53]	✓			✓			✓	✓	

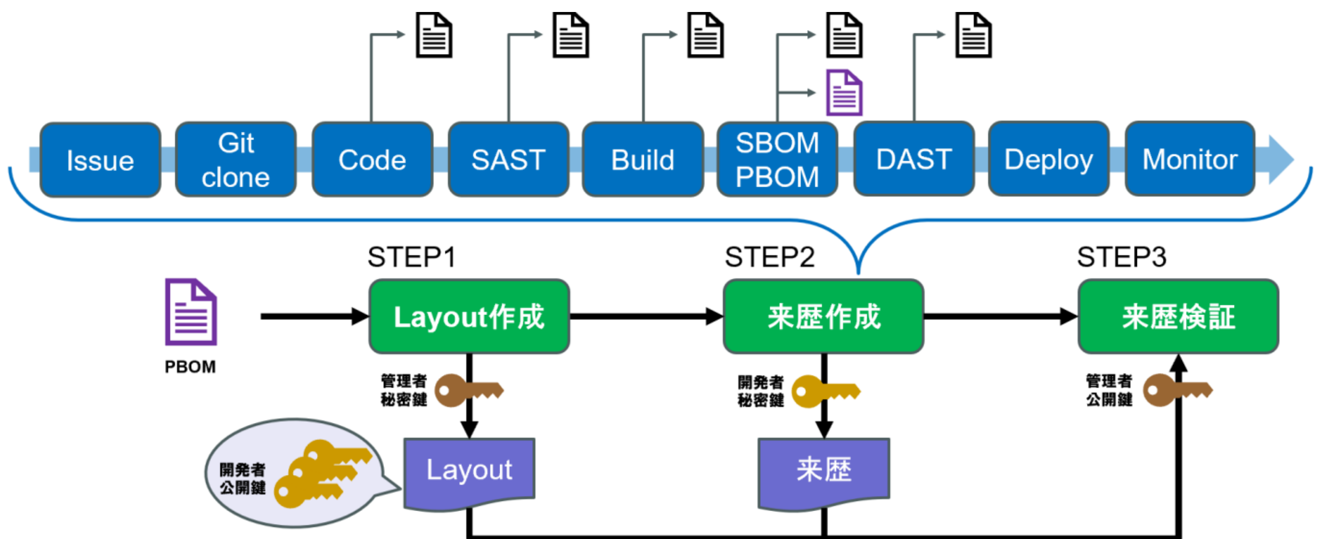

図 3　DevSecOps パイプライン全体の来歴検証を可能とする提案手法の全体像

4.3 PBOM（Pipeline Bill of Materials）

PBOM は，パイプラインを構成する全ての要素（ツール，スクリプト，ライブラリ，サービス等）を記述した情報リストであり，SBOM の概念をパイプラインに適用したものである．SBOM が開発成果物（アーティファクト）に含まれるソフトウェアコンポーネントの一覧を管理するのに対し，PBOM はパイプラインに使用される構成要素の一覧を管理対象とする．

PBOM の概念は，OX Security 社[6]によって示されているが，その具体的な要素や管理方法については言及されていない．先に挙げたガイドラインにおいても構成要素や，生成方法については明確に定義されておらず，PBOM の定義や生成方法は標準化されていないのが現状である．

本研究では，PBOM の構成要素を整理し，PBOM 生成ツールを開発する．さらに，生成された PBOM を in-toto による来歴検証プロセスに活用することを目指す．

5. 提案手法

5.1 提案手法の全体像

DevSecOps パイプライン全体の SSC 可視化と来歴検証を実現するために，PBOM の自動生成と，in-toto を用いた来歴検証を組み合わせた手法を提案する（図 3）．

本手法は以下 2 つの機能を実現するものである：

① パイプライン解析による PBOM の生成

② in-toto の来歴検証によるパイプライン全体に対する SSC 検証

5.2 パイプライン解析による PBOM の生成

本研究では，CI/CD パイプラインの構成を記述する YAML ファイルを入力とし，SBOM と同等の形式で PBOM をアーティファクトとして出力する PBOM 生成ツールを開発する．PBOM フォーマットとして SBOM の代表的な形

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

式である JSON 形式に対応する他，インプットファイルと同形式である YAML への対応を前提とする．

解析対象とする CI/CD パイプラインツールは，Okafor らの研究[3]でも挙げられている GitHub Actions[25]を主な対象とするが，GitLab CI/CD[26]や，AWS CodePipeline[27]といった代表的な CI/CD ツールへの対応も視野に入れている．

YAML ファイルによるパイプライン解析では，YAML 構文からステップ名や実行ツール名等の必要情報を抽出し，出力フォーマットに沿った形式に整理する方法としている．

5.3 in-toto の来歴検証によるパイプライン全体に対する SSC 検証

in-toto には Layout 作成，来歴作成，来歴検証という 3 つの基本機能が備わっている．本章では，それぞれの機能についての説明，および DevSecOps におけるパイプライン検証への適用方法について述べる．

5.3.1 Layout 作成

Layout とは in-toto による来歴検証において，検証元となるホワイトリストの役割を持つファイルである． Layout は，プロセス毎の要件（実行ユーザーの公開鍵，実行コマンド，アーティファクト名など）を定義し，Layout 作成者の秘密鍵によって署名される．

提案手法では，この Layout に DevSecOps パイプライン全てのプロセスを定義することで，パイプライン全体の SSC を検証することを目指している．

Layout 作成の工程は，SSC の整合性を確保する上で重要となるプロセスである．Layout の定義に誤りや漏れがある場合，期待される検証が正しく行えず，信頼性が損なわれる恐れがある．

しかし，Layout 作成機能は CLI として提供されていないため，in-toto で定義された Layout 向けの Python ライブラリを使用して作成する必要がある．開発者が Python スクリプトを手動で作成し，各ステップの定義をスクリプト内に直接記述する必要があるため，作成コストが高く，ヒューマンエラー（記述漏れや定義ミス）が発生しやすい状況である．

これらの課題に対応するため，提案手法では 5.2 節で述べた PBOM を入力情報として Layout ファイルを作成することで， Layout ファイル作成のコスト削減，および品質向上の実現を図る．

5.3.2 来歴作成

来歴（Provenance）とは，Layout で定義したプロセスの実行内容を記録したリンクメタデータである．「in_toto_run」API を介してプロセス実行することで来歴が作成される．来歴には，実際に実行コマンドや，出力されたアーティファクト等の情報を含み，実行ユーザーの秘密鍵によって署名が行われる．これらの情報を元に，5.3.1 で作成した Layout ファイルと比較することで，事前に定義したプロセスの要件を満たしているかの検証が可能となる．

5.3.3 来歴検証

5.3.1 および 5.3.2 節で述べた Layout と来歴を比較検証する機能である．「in_toto_verify」API を介して Layout ファイル毎に検証を実施する形を取る． 1 つの Layout に複数のプロセスが定義可能であるため，全てのプロセスを 1 つの Layout に集約した場合は，1 度の検証で全てのプロセスについて確認することが可能である．

来歴検証は CI/CD パイプライン処理の最終段階，すなわちデプロイ直前に実施することを想定している．デプロイ直前に，それまでに経た全てのプロセスを検証することで，リリース直前のセキュリティゲートとしての機能が期待できる．

しかし，本研究では CI/CD パイプラインに含まれない DevSecOps プロセス（Issue 管理や Git 操作，モニタリング等）についても検証対象とすることを目指しており，検証のタイミングや検証単位（分割処理の可否）については検討の余地がある．

5.4 本研究の貢献

本研究では,DevSecOps パイプライン全体における SSC の来歴検証を実現するための新たなアプローチを提案することで，以下 3 点の貢献を期待している：

① CNCF が提唱するベストプラクティスに基づいた DevSecOps パイプライン全体を対象とした，具体的な SSC 検証手法の確立
② PBOM によるパイプライン管理手法の確立
③ PBOM と in-toto を組み合わせた SSC 検証の自動化による検証コスト削減，および品質向上への貢献

5.4.1 具体的な SSC 検証手法の確立（貢献①）

提案手法によって実現される DevSecOps パイプライン全体の SSC 来歴検証は，SLSA モデルでは対応していない脅威ポイントに対しても効果が期待できると考えている．

対応可能となる脅威ポイントを図 4 に示す．SLSA では A～H の脅威ポイントが定義され，A～G の脅威ポイントについては高レベルの SLSA モデルを構築することで対処可能であるとし，H の脅威ポイントについては SLSA で直接対処することはできないとしている．例として，Orion の事例はビルドプラットフォームが侵害されたことが原因であるため，E の脅威に位置付けられており，SLSA モデルに基づいてビルド環境をハードニングすることで対処可能としている．

提案手法においては，DevSecOps における全てのプロセスを対象とするため，H～N の脅威ポイントについて対処可能となることが期待できる． 例として，Orion の事例をユーザー視点で考えた場合，モニタリングツールを使用する M の脅威ポイントにおいて，ユーザー側で意図しない挙動を検知できることが期待できる．

5.4.2 PBOM によるパイプライン管理手法の確立（貢献②）

4.3 節で述べた通り，PBOM の概念は提案されているが，その具体的な要素は明確に定義されていない．

そこで，本提案手法では PBOM の具体的要素を検討した．さらに，SSC 検証の自動化（5.4.3）への活用を目指している．本稿執筆時点では，自動化の実装と合わせて定義化を進めている状況であるが．SBOM の要素（サプライヤー名，コンポーネント名，バージョン情報，依存関係，等）に加えて，パイプラインの独自情報を追加することを検討した．

例として，パイプラインのどのプロセスで利用されているかのステップ情報や，実行コマンド，実行オプションなどを含む管理対象の操作に関する情報の追加をした．これにより，in-toto による来歴検証の自動化への活用が期待できる．

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

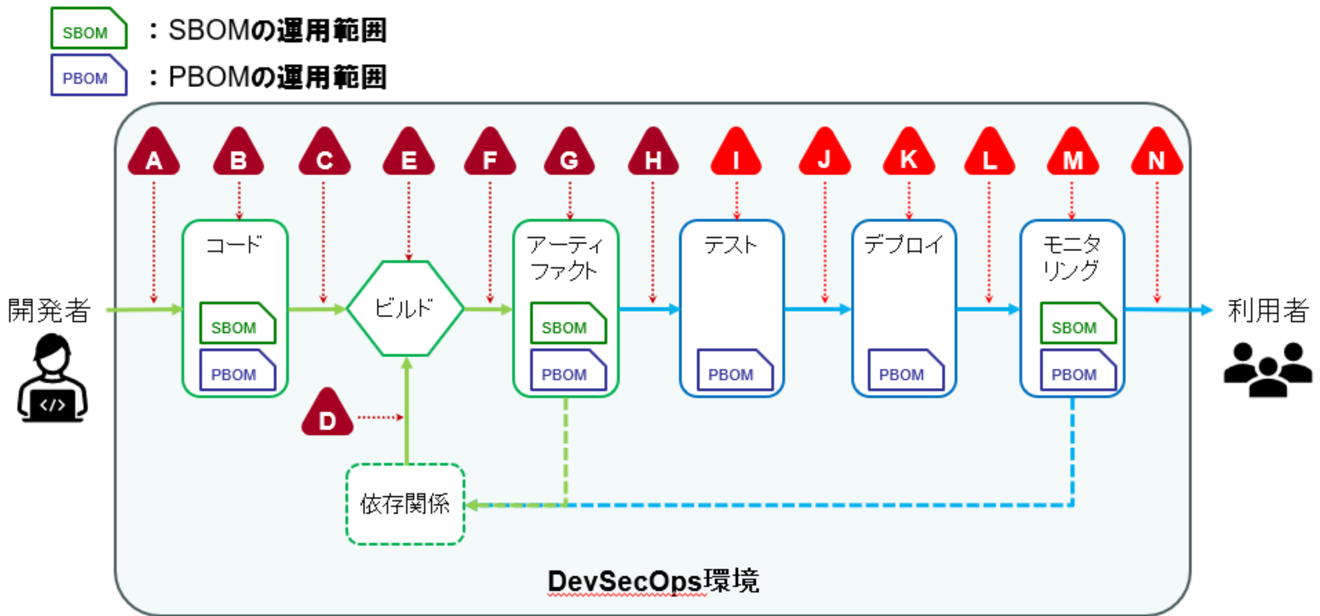

図 4　　提案手法により対応可能となる脅威ポイント

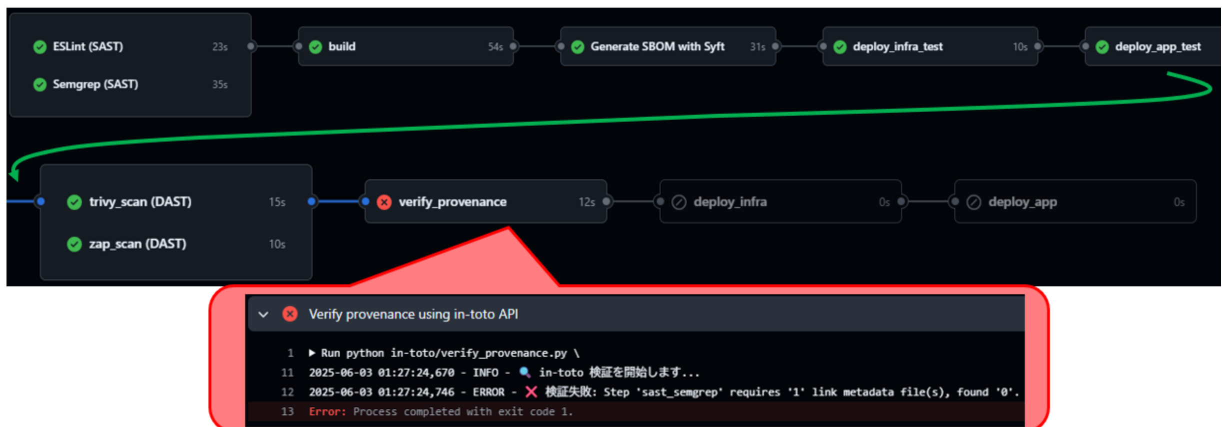

図 5　　来歴検証の結果で本番環境へのデプロイを制御

5.4.3 SSC 検証の自動化（貢献③）

貢献①，および貢献②については，任意のタイミングで開発者自身が手動実行することで実現可能が可能であるが，手動実行による管理・検証では，DevSecOps が求めるスピーディーな開発の実現は困難である．また，ヒューマンエラーによる検証の漏れ・不備が発生するリスクが生じてしまう．そこで，これらの提案手法を DevSecOps パイプライン上で自動化する方法を提案する．

PBOM の生成については，SBOM 生成ツールと同様に CI/CD パイプライン内の処理として PBOM 生成ツールの実行を組み込むことで自動化を実現する．また，システムリリースを高頻度で実施する DevSecOps の特性を考えると，SBOM，PBOM も高頻度で更新されるため，SBOM の課題で挙げられているクラウドサービスの頻繁な更新に対応可能となることも期待できる．

さらに，生成した PBOM をインプットファイルとして in-toto の Layout 作成を実現する機能を開発することで，来歴による SSC 検証の自動化を実現することを目指している．

なお，Layout 作成において，提案手法が構築された 2 回目以降の作成については自動化で対応されるが， PBOM が作成されていない初回の Layout 作成方法については考慮が必要である．例として，検証環境へのリリースから本番環境へのリリースまで連続したパイプラインなのか，環境毎にパイプラインが分割されているのかの違いによって初回の PBOM の生成方法や管理単位の在り方が変わると考えている．この課題については，実装を進めると同時に，より実用的な方法を目指す．

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

6. 提案手法の実装と課題

6.1 開発環境

本提案手法の実装で使用している環境は以下の通り．

- 開発言語：Python 3.12.0
- 主なライブラリ：in-toto 3.0.0，securesystemslib 1.3.0
- CI/CD ツール：GitHub Actions
- デプロイ環境：AWS（CloudFormation）

6.2 提案手法の実装

5.1 節で述べた 2 つの機能について，本稿執筆時点での実装状況について述べる．

6.2.1 機能①：パイプライン解析による PBOM の生成

GitHub Actions の CI/CD パイプライン（YAML ファイル）を解析し，PBOM を生成する Python スクリプトを実装した．PBOM の要素として最低限必要であると考えている，ステップ情報，ツール名，バージョン情報，実行コマンドを抽出することができている．この PBOM を元に in-toto の Layout 自動生成を進めている段階であり，実行を許可するユーザー情報の定義方法や，ビルドスクリプトのような独自スクリプト内で使用されるツールの管理（再帰性）の実現などの改良を検討している．

6.2.2 機能②： in-toto の来歴検証によるパイプライン全体に対する SSC 検証

図 3 で示した in-toto の基本機能である Layout 作成，来歴作成，来歴検証を実現する Python スクリプトを実装した．

来歴作成と来歴検証のスクリプトについては CI/CD パイプラインの処理として組み込み，SSC プロセスに問題があった場合には，本番環境へのデプロイを中止するセキュリティガードとして機能することが確認できた．図 5 は，何らかの理由で SAST 処理実行時の来歴が作成されていないために，来歴検証で異常が検知され，本番環境へのデプロイを中止したことを示している．

CI/CD として実施される DevSecOps プロセス（SAST，Build，SBOM・PBOM 生成，DAST）を対象とした来歴検証は実現できているが，CI/CD 外のプロセスに対する来歴検証は現時点で自動化できていない．CI/CD 以前のプロセスについては，来歴情報を CI/CD に渡すことで同じ検証プロセスで検証可能と考えている．CI/CD 以後のプロセスについては，別の検証プロセスとして追加を検討しており，全体のプロセスを関連付けするために CI/CD で出力されたアーティファクトを入力情報として Layout に定義することで全体の検証を実現したいと考えている．

また，CI/CD パイプライン処理の実行ユーザー識別は GitHub ユーザーで認識している．in-toto では実行者の秘密鍵を使用して来歴作成を実施する必要があり，秘密鍵は GitHub リポジトリのシークレット情報として事前登録して対応している．

6.3 実装上の課題

6.2 で示した実装において，来歴検証を行う「in_toto_run」API を通じて処理を実行することで，正常に動作しない処理が確認されている．具体的には，自前のシェルスクリプトの実行は正常に動作するが，各ツールが用意している CLI を利用する場合に正常に動作しない場合がある．原因究明のために詳細な調査を進めているが，特定のコマンド構造やコンテナを使用した処理について工夫が必要であることが想定され，今後，解決を図る．

7. 今後の展望

本研究の今後の展望として，次の 3 点を挙げる．

7.1 提案手法の自動化範囲拡大

本研究では，PBOM を活用した DevSecOps パイプライン全てのプロセスを対象とした来歴検証の自動化を目指しているが，現時点で全てを自動化するに至っていない．そのため，図 3 で示したフレームワークの全体像に沿って，自動化範囲の拡大を実現したいと考えている．特に，CI/CD 外のプロセスに対して，DevSecOps で実行されるプロセスについて，対応範囲を広げることが望ましいと考えている．

7.2 ユーザー認証方法の改善

6.2.2 節で述べた通り，CI/CD パイプライン内の処理については GitHub リポジトリのシークレット情報を用いて認証を実施しており，実際に企業・組織で利用する場合には，強固な認証基盤との連携が必要であると考えている．具体的には，GitHub アカウントが守られている前提のフレームワークとなっているため， GitHub アカウントが乗っ取られた場合や，シークレット情報が搾取された場合の脅威に対応できていない．

in-toto による来歴検証では，ユーザーの秘密鍵による署名が必要なため，鍵の管理が重要である．つまり，KMS（Key Management Service)や，TPM （Trusted Platform Module）を利用した安全な鍵の運用方法を導入する案が考えられる．

7.3 再帰的なパイプライン検証の実現

本稿では，DevSecOps パイプラインの SSC 検証手法を示した．これにより，自組織で構築した DevSecOps パイプラインを自組織で検証することが可能になると考える．なお，検証結果を外部組織が検証する方法の実装など残課題もあるが，これについては検証元となる PBOM および検証結果を外部組織に提供することで，サプライチェーンにおいて再帰的な検証が実現できると考えている．

PBOM の提供については，SBOM と同様にサプライチェーン間で共有することで，開発アーティファクトの透明性が向上すると考えているが，in-toto による来歴検証の特性上，具体的な実行コマンドや実行許可ユーザーの情報等の機微な情報を含むため，PBOM の構造変更や，情報提供範囲の検討が必要である．

検証結果の提供については，検証結果をアーティファクトとして電子署名を付与することで検証結果を証明できると考えている．具体的にはアーティファクトへの署名フレームワーク（sigstore[24]）を統合することで実現することを考えている．

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.

8. まとめ

本稿では，DevSecOps 環境における SSC の脅威を示すとともに，関連ガイドラインおよびフレームワークの現状を述べた．そして，DevSecOps パイプライン全体を対象範囲とした SSC 検証を実現する包括的なフレームワークが確立されていないことに着目し，SBOM の概念を拡張した PBOM による SSC 管理手法と，in-toto による来歴検証を拡張することで DevSecOps パイプライン全体の SSC 検証を実現する手法を提案した．

提案手法の実装においては，PBOM 生成ツールを実装し，in-toto の Layout に活用するための情報を含めた PBOM を生成することで，自動化に向けた方針を示した．また，in-toto の基本機能を実現するスクリプトを実装し，SAST，SBOM・PBOM 生成，DAST などのセキュリティプロセスを統合した CI/CD パイプラインの SSC 検証を示した．

一方で，in-totoAPI を介して実行する際に個別対応を要する処理や，CI/CD パイプライン外の DevSecOps プロセスに対する来歴検証の自動化には課題を残していることを述べた．

今後の展望としては，提案手法の自動化範囲拡大，ユーザー認証方法の改善，再帰的なパイプライン検証の 3 点について述べ，それぞれ今後の方針を示した．

参考文献

[1] OWASP: DevSecOps Guideline - v-0.2.
[2] Carol Woody, Tim Chick, Aaron Reffett, Scott Pavetti, Richard Laughlin, Brent Frye, Mike Bandor: DevSecOps Pipeline for Complex Software-Intensive Systems: Addressing Cybersecurity Challenges, Journal of Systemics, Cybernetics and Informatics (2020).
[3] Chinenye Okafor, Taylor R. Schorlemmer, Santiago Torres-Arias, James C. Davis: SoK: Analysis of Software Supply Chain Security by Establishing Secure Design Properties, SCORED ’22, November 11, 2022, Los Angeles, CA, USA(2022).
[4] 経済産業省: ソフトウェア管理に向けた SBOM（Software Bill of Materials）の導入に関する手引 ver 2.0.
[5] CISA: Software Transparency in SaaS Environments.
[6] Ox Security: The Anatomy of a PBOM, https://www.ox.security/the-anatomy-of-a-pbom/, Accessed 2025-6-10.
[7] in-toto: https://in-toto.io/ Accessed 2025-1-8.
[8] Linux Foundation: Cloud Native 2024.
[9] SHIFT: DevSecOps とは？ソフトウェアの開発・運用・セキュリティ対策を ONE TEAM で https://service.shiftinc.jp/column/8337/, Accessed 2025-6-3.
[10] Gartner Research: DevOpsSec: Creating the Agile Triangle , https://www.gartner.com/en/documents/1896617, Accessed 2024-10-21.
[11] GitLab: 2024 グローバル DevSecOps レポート, https://about.gitlab.com/ja-jp/developer-survey, Accessed 2024-09-12.
[12] Xiaofan Zhao, Tony Clear, Ramesh Lal: Identifying the primary dimensions of DevSecOps: A multi-vocal literature review, Journal of Systems and Software Volume 214(2024).
[13] IPA：情報セキュリティ 10 大脅威 2025.
[14] GAO Cybersecurity: Federal Response to SolarWinds and Microsoft Exchange Incidents , https://www.gao.gov/products/gao-22-104746, Accessed 2024-10-21.
[15] Enisa: ENISA THREAT LANDSCAPE FOR SUPPLY CHAIN ATTACKS(2021).
[16] 唐沢勇輔: 「Log4shell」は何故これだけ騒がれたのか, https://www.ipa.go.jp/security/sc3/activities/kougekiWG/content/column/col-vol02.html, Accessed 2024-10-21.
[17] 米国大統領令 14028: https://www.whitehouse.gov/wp-content/uploads/2022/09/M-22-18.pdf, Accessed 2025-6-2
[18] NIST: NIST SP 800-218 Secure Software Development Framework (SSDF) Version 1.1: Recommendations for Mitigating the Risk of Software Vulnerabilities.
[19] NIST: NIST SP 800-204D Strategies for the Integration of Software Supply Chain Security in DevSecOps CI/CD Pipelines.
[20] Mehdi Mirakhorli, Derek Garcia, Schuyler Dillon, Kevin Laporte, Matthew Morrison, Henry Lu, Viktoria Koscinski, Christopher Enoch : A Landscape Study of Open Source and Proprietary Tools for Software Bill of Materials(SBOM), arXiv (Cornell University)(2024)
[21] CIS: CIS Software Supply Chain Security Guide v1.0.
[22] SLSA: SLSA v1.1.
[23] CNCF: Software Supply Chain Best Practices v2.
[24] Sigstore: https://www.sigstore.dev/, Accessed 2025-6-3.
[25] GitHub: https://docs.github.com/ja/actions, Accessed 2025-6-3.
[26] GitLab: https://docs.gitlab.com/ci/, Accessed 2025-6-3
[27] AWS: https://aws.amazon.com/jp/codepipeline/, Accessed 2025-6-3.

Copyright © 2025 by
The Institute of Electronics, Information and Communication Engineers and
Information Processing Society of Japan All rights reserved.