SBOMと実行されるプログラムの不一致に対処するアクセス制御手法の提案

島本 裕大[1,a)]　上川 先之[2]　秋山 満昭[2]　山内 利宏[3]

概要：複数のソフトウェアが混在する実行環境やファイルシステムでは，すべてのソフトウェア構成情報を正確に把握することは困難である．これを明示的に示す手段としてSBOMがあり，ソフトウェア構成の可視化やこれによる既知の脆弱性の検知ができる．しかし，SBOMは実際のソフトウェア構成と一致しない可能性がある．これは，SBOMが最新の構成を常に反映するものではなく，特定のタイミングで作成される静的な情報であるためである．また，SBOM作成時に，リスクが生じる可能性のある全てのプログラムの正確な情報が網羅されるとは限らない．特に，SBOMと実際に実行されるプログラムの間に不一致がある場合，発生しうる脅威を見逃すリスクが高まる．このため，両者の一貫性を保つことが重要である．本稿では，SBOMと実際に動作するプログラムの情報の不一致に対処するアクセス制御手法（SBOM-AC）を提案する．SBOM-ACはアクセス制御により，SBOMの情報と一致しない実行されようとするプログラムに対して，実行拒否やファイル情報の記録を行う．SBOM-ACの設計と実現方法について述べ，プロトタイプを用いて検証し，不一致時に拒否と記録が行えることを示した結果と考察について述べる．

キーワード：SBOM，アクセス制御，ソフトウェアサプライチェーン

SBOM-based Access Control Against Executed Programs Discrepancies

Yuta Shimamoto[1,a)]　Hiroyuki Uekawa[2]　Mitsuaki Akiyama[2]　Toshihiro Yamauchi[3]

Abstract: Accurately identifying all software in environments with multiple programs, like execution environments or file systems, is challenging. SBOM provides this information, enabling visualization of software composition and detection of known vulnerabilities. However, an SBOM may not accurately reflect the actual software information. SBOM does not update the latest composition information itself. In addition, when creating SBOMs, not all programs that may pose a risk are always included. In particular, when there is a discrepancy between SBOMs and the programs that are executed, we might miss potential threats. Therefore, maintaining consistency between those two is important. In this paper, we propose SBOM-AC to handle discrepancies as they arise. SBOM-AC uses access control to deny execution or record file information of programs about to be executed that do not match the information in the SBOM. Using a prototype, we demonstrated that refusal and recording can be performed when discrepancies occur.

1. はじめに

Software Bill of Materials（SBOM）は，ソフトウェアを構成するコンポーネントの情報を構造化して記述するフォーマットである．コンポーネントの利用者は，これの構成情報を明示的に示したSBOMを確認することで，ソフトウェアの構成要素とライセンスの情報を把握できる．これにより，ソフトウェアの中身を解析することなく，SBOM

[1] 岡山大学大学院環境生命自然科学研究科
Graduate School of Environmental, Life, Natural Science and Technology, Okayama University
[2] NTT 社会情報研究所
NTT Social Informatics Laboratories
[3] 岡山大学学術研究院環境生命自然科学学域
Faculty of Environmental, Life, Natural Science and Technology, Okayama University
[a)] shimamoto@s.okayama-u.ac.jp

に基づいた既知の脆弱性検知やライセンス管理ができる．

一方で，ソフトウェアの開発や運用の過程で構成に変更があったとしても，その変化を自動的に SBOM へ反映することはできない．これは，SBOM は通常，特定のタイミングで静的に作成されるためである．このため，SBOM に記載された構成情報との一貫性が失われる場合が考えられる．例えば，開発時に依存ライブラリのバージョンが変更された場合，および運用時にパッチの適用や外部からのプログラムのダウンロードが行われた場合が挙げられる．これに加え，SBOM の作成者が悪意を持って一部の情報を意図的に記載しない可能性もある．このような状況では，現状の SBOM を用いた脆弱性管理やライセンス監視には限界があり，SBOM に記載されていない構成要素に起因するリスクを正確に把握することが難しい．実際のソフトウェア構成に含まれているにもかかわらず，SBOM に記載されていないプログラムが既知の脆弱性を持っていたり，悪意のある動作をする可能性がある．このような場合，これらのリスクを SBOM から把握することはできない．

SBOM-based Access Control（以降，SBOM-AC）は，SBOM と実行されるプログラムの情報の一貫性が崩れることよって発生する問題を緩和する．具体的には，SBOM の情報と一致しない実行されようとするプログラムを検知して対処する．SBOM-AC は，不一致を検知したプログラムへの対処として，ファイル情報の記録と実行拒否の 2 つの機能を持つ．製品の開発中では，検知したプログラムをログとして記録をすることで，不一致のプログラムの存在を可視化する．これらのプログラムを調査し，問題があれば適切に対処，問題がなければ SBOM に含めることで，よりセキュアな製品と脆弱性管理に有用な SBOM を作成できる．また，製品のリリース後では，ファイル情報の記録に加えて実行拒否を行うことで，SBOM と整合性を持ったプログラムのみ実行できるようにする．これにより，SBOM との不整合により確認できない，悪意のあるプログラム実行などの被害を防止できる．

本稿では，SBOM-AC の設計方針について示し，実現方法と対処の内容について述べる．次に，想定されるユースケースにおける SBOM-AC の有用性を示す．さらに，SBOM-AC のプロトタイプを実装し，検証した結果を報告する．具体的には，SBOM の内容と一致しない実行されようとしているプログラムの検知，およびファイル情報の記録と実行拒否がそれぞれ可能か確認した．

以下に，本研究の主な貢献を示す．

(1) 今後ますます普及が見込まれる SBOM からアクセス制御ポリシーを生成し，強制アクセス制御を行う SBOM ベースのアクセス制御手法を初めて提案した．
(2) SBOM の利活用のユースケースを想定し，意図せずに SBOM に含まれていないプログラムを検知できることや，SBOM 生成後に悪意のあるプログラム等を追加された場合の実行防止機能などが実現可能な SBOM-AC の設計を示した．
(3) 設計に基づくプロトタイプ実装による検証により，SBOM-AC の実現可能性を示した．

2. 背景

2.1 SBOM

SBOM は，ソフトウェアパッケージとそのコンテンツを一意に識別する形式的で機械可読なメタデータである．これを用いることで，ソフトウェアサプライチェーンの関係者に提供されるコンポーネントの情報を共有できる [1]．

実行環境やファイルシステム（以降，システム）には，複数のソフトウェアが混在する．代表的な IoT 機器であるホームルータのファームウェアには，平均 662 のコンポーネントが含まれる [2]．このため，すべてのソフトウェアの情報を把握することは難しい．SBOM を用いることで，中身を解析することなくソフトウェア構成を把握できる．

SBOM は，各国で導入が推進されており，今後システムに対して標準で付随するデータとなることが予想される．2021 年 5 月に発出された米国大統領令（EO14028） [3] には，ソフトウェアサプライチェーンの強化が含まれており，連保政府機関に納入されるソフトウェアに対してベンダや開発者が SBOM を作成することを求めている [3]．これを受け，米国電気通信情報庁（NTIA）とサイバーセキュリティ・インフラセキュリティ庁（CISA）が SBOM のガイドライン作成を進めている．EU ではサイバーレジリエンス法（以降，CRA）でデジタル要素を備えた全ての製品に対して SBOM の作成を求めている [4]．また日本では，経済産業省がソフトウェア管理に向けた SBOM の導入に関する手引 [5] を公開している．

SBOM には, コンポーネントの名前, バージョン, SBOM の作成者，他のコンポーネントとの依存関係，及びライセンス情報などが明示的に記載される．これらの情報を用いて，脆弱性管理やライセンス管理が可能となる．

2.2 SBOM の作成方法

CISA が 2024 年 9 月に公開した SBOM についての資料 [6] に基づいて，ソフトウェアコンポーネントの SBOM の作成方法について説明する．

SBOM を作成するのは，サプライヤである．サプライヤは，ソフトウェアコンポーネントの開発者または製造者を指す．ただし，コンポーネントとこれの SBOM の作成者が同一とは限らない．

SBOM を作成するときは，対象コンポーネントに含まれる依存コンポーネントの SBOM を可能な限り収集する．SBOM が入手できない場合は，可能な限り独自で作成する．これにより収集と作成した SBOM の情報を全て含めるように対象コンポーネントの SBOM を作成する．この

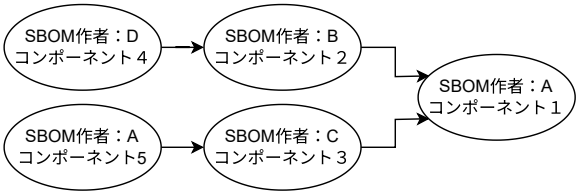

図 1 依存関係の SBOM 概念図

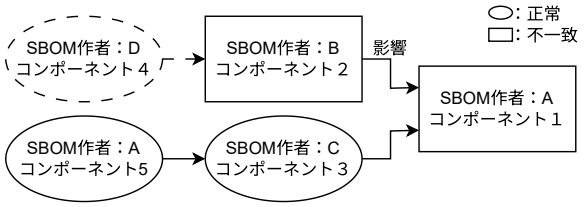

図 2 実際の構成に存在するコンポーネントが SBOM に記載されない脅威

とき作成する SBOM は，ベストエフォートでの作成が望まれている．すなわち，完璧なものである必要はない．

SBOM 作成の例として，**図 1** を示す．A がコンポーネント 1 の SBOM を作成する例で説明する．コンポーネント 1 は，コンポーネント 2 と 3 に依存しており，コンポーネント 2 はコンポーネント 4，コンポーネント 3 はコンポーネント 5 に依存している．このとき，A はコンポーネント 2 と 3 の SBOM は取得できたものの，コンポーネント 3 にコンポーネント 5 の情報が含まれていなかった．このため，コンポーネント 5 の SBOM は自作する必要がある．これにより，収集と作成をしたコンポーネントの SBOM 全ての情報を含めて，コンポーネント 1 の SBOM を A が作成する．

製品の SBOM の場合，少なくとも製品のトップレベルの依存関係をカバーするよう，CRA のドキュメントに書かれている [7].

2.3 SBOM を用いた脆弱性管理方法

CISA の資料によると，SBOM を用いた脆弱性管理方法として，CVE，NVD，および VEX を使う方法がある [6].

CVE と NVD からは既知の脆弱性情報を取得できる．コンポーネントと公開されている脆弱性情報を照合することで，脆弱性の有無を知ることができる．SBOM を用いることで，コンポーネントに含まれている照合に必要な情報が明示的にわかる．このため，コンポーネントの既知の脆弱性の迅速な調査が可能となる．

サプライチェーン上流の一部のコンポーネントは，一部使用されない場合がある．コンポーネントがライブラリの場合，これの一部しか呼び出されない場合や，コンポーネントの特定の機能がビルド時やパッケージング時に無効化されることがある．このため，SBOM に含まれる全てのコンポーネントの脆弱性が製品に影響するわけではない．

VEX は，コンポーネント内の脆弱性の状態（影響なし，影響あり，修正済み，調査中）を示す．これにより，コンポーネント内部に脆弱性が見つかったとしても，この脆弱性がコンポーネントに影響するかを示すことができる．サプライヤとユーザは，影響を受けていないコンポーネントのセキュリティ更新プログラムの管理，作成，および適用にかかるコストを削減できる．

3. 脅威モデル

3.1 脅威モデルの概要

SBOM を用いた脆弱性管理は迅速に行える．これはコンポーネントに含まれているプログラムの情報を SBOM から確認できるためである．しかし，これは SBOM と実際のソフトウェア構成が一致していることが前提である．

SBOM に記載されている状況と実際の構成で不一致が生じる状況がある．これは，発生し得る脅威の見逃しや誤検知につながる．SBOM と実際の構成を常に一致させることで，SBOM を使用した脆弱性管理の効果を上げ，製品をよりセキュアにできる．

不一致が生じるタイミングは，リリース前の SBOM 作成時，およびリリース後のユーザ使用時に分けられる．以降の節では，それぞれについて，発生する状況，不一致の発生要因となる攻撃者または関与者，および被害者を示す．

3.2 リリース前の SBOM 作成時の脅威モデル

情報が十分に含まれていない上流の SBOM を信頼して製品の SBOM を作成すると，脆弱性やマルウェアがコンポーネントに含まれていることが明記されない．これは，SBOM を使用した脆弱性管理で見逃しが発生し，脅威の発見と対処が遅れる原因になりうる．このとき，サプライチェーン上流の SBOM 作成者は攻撃者となる．また，被害者は製品の SBOM の作成者である．

この脅威の例を**図 2** に示す．入手したコンポーネント 2 の SBOM に実際の構成では含まれているコンポーネント 4 の情報が含まれていないため，この情報をもとに作成したコンポーネント 1 の SBOM にも不一致が生じている．

また，古い情報の SBOM を用いて製品の SBOM を作成すると，脆弱性管理において問題が生じる可能性がある．具体的には，すでに対処済みの脆弱性を誤って検知してしまったり，新たに発生した脆弱性を見逃してしまう．古い情報の SBOM を用いてしまう原因として，サプライチェーン上流の SBOM 作成者がシステムの更新に合わせて SBOM を更新しないことや SBOM の作成者自身が更新の確認を怠ることがある．このため，関与者はサプライチェーン上流の SBOM 作成者と製品の SBOM 作成者自身となる．また，被害者は製品の SBOM 作成者である．

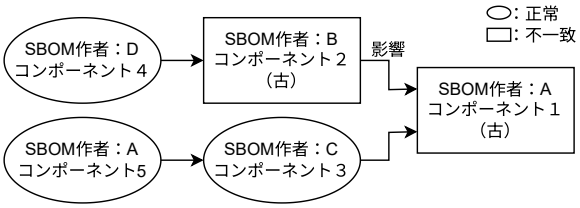

図 3　コンポーネントの最新の構成が SBOM に記載されない脅威

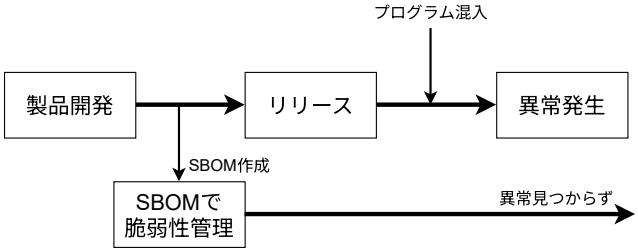

図 4　SBOM 作成後にプログラムが混入した場合対処できない脅威

この脅威の例を**図 3** に示す．入手したコンポーネント 2 の SBOM が古いため，この情報をもとに作成したコンポーネント 1 の SBOM にも古い情報が記されている．結果，実際の構成との不一致が生じる．

3.3　リリース後のユーザ使用時の脅威モデル

製品のリリース後にプログラムが混入した場合，SBOM はこれを反映することができない場合の脅威例を**図 4** に示す．リリース前に正確な SBOM が作成できていたとしても，リリース後の製品にプログラムの混入があった場合 SBOM はこの内容を反映できない．このため，SBOM を用いた脆弱性管理で異常を発見できない．

このときの攻撃者は，製品にアクセスして不正なプログラムの設置とこれを実行する能力を持つ．ただし，製品にアクセスして正規のプログラムを悪用することについて今回は対象としない．これは，SBOM との不一致とは関係がなく対処したい目的とは異なるためである．被害者は，製品を使用するユーザである．混入したプログラムが実行されることにより，所持している製品の悪用や情報が窃取がされる．

4.　提案手法とユースケース

4.1　提案手法

SBOM-AC は，実行されようとするプログラムを一意に特定できる情報を SBOM から取り出して作成したポリシーをもとに，強制アクセス制御（以降，MAC）を行う．これにより，SBOM から隠されたプログラムのロギングや実行拒否を行い，SBOM に記載されている情報と実際に実行されるプログラムの不一致に対処する．

実行されようとするプログラムを一意に特定できる情報として，ファイルパスとハッシュ値の組を使用する．ハッシュ値を含めているのは，既存のプログラムに上書きされる形で悪意のあるプログラムが設置される可能性があるためである．これにより，実行されようとするプログラムに対して，以下の 2 種類の不一致を検知する．

(1) SBOM に記載されていない
(2) SBOM の情報とハッシュ値が異なる

これにより，SBOM への未記載，およびリリース後に攻撃者が設置したプログラムの実行に対処する．

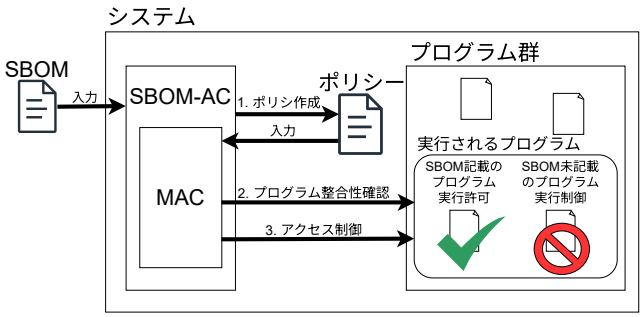

図 5　SBOM-AC の概要

SBOM-AC の概要図を**図 5** に示す．SBOM-AC は，ポリシー生成，プログラム整合性確認，およびアクセス制御の 3 つの要素で構成される．ポリシー生成では，SBOM からファイルパスとハッシュ値の組を抽出し，この情報に一致するプログラムの実行を許可するポリシーを作成する．プログラム整合性確認では，実行されようとするプログラムの情報が SBOM と一致するかポリシーの内容に基づき判断する．アクセス制御では，不一致のプログラムに対して状況に応じてロギングや実行拒否で対処する．

ロギングは，不一致のプログラムのパスをログに残す．これにより，プログラムの実行に影響を与えずにどのプログラムで不一致が発生したのかわかる．実行拒否は，不一致のプログラムの実行を拒否する．これにより，SBOM を用いた脆弱性管理で調査できなかったプログラムが実行されなくなり，安全である．

SBOM-AC のアクセス制御は，MAC として実現する．MAC であるため，管理者権限のプログラムであっても，SBOM-AC のポリシーで実行可否を制御する．SBOM-AC のポリシーは，SBOM から生成し，ポリシーに記載されているファイルパスとハッシュ値に一致するプログラムのみ実行を許可し，これ以外の場合は状況に応じてロギングや実行拒否で対処する．

SBOM-AC のユースケースは開発中とリリース後の 2 つある．それぞれ対処できる脅威や望ましい対処方法が異なる．以降の節では，これらについてそれぞれ示す．

4.2　開発中におけるユースケース

製品の開発中は，SBOM-AC の機能を有効にしてテストを行う．これにより，外部のコンポーネントに含まれる悪意のあるプログラムが SBOM から隠されている状況で，これが実行された場合に対処可能である．このため， 3.2 節

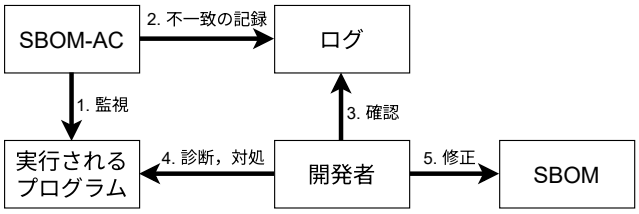

図 6 開発中における SBOM-AC のユースケース

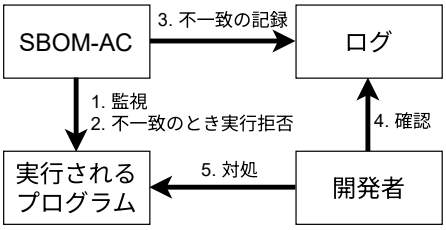

図 7 リリース後における SBOM-AC のユースケース

で示したリリース前の製品作成時の脅威による見逃しに対処できる．

このとき，SBOM-AC が行う対処は，ロギングのみであるのが望ましい．実行を拒否すると，SBOM と不一致のプログラムを網羅的に検知できない可能性がある．例えば，実行拒否したプログラムが別のプログラムを実行する構造だった場合，このプログラムは検知されない．ロギングのみをすることで，網羅的に SBOM と不一致なプログラムを検知できる．

また，このユースケースでは，実行されるプログラムの意図的でない不一致にも対処できる．意図的でない不一致の原因として，ヒューマンエラーや SBOM 作成ツールで作成した SBOM の不確実性 [8] が挙げられる．ロギングにより確認できる情報をもとに SBOM を修正することで，より正確性の高い SBOM を作成できる．

このユースケースが脅威をどのように解決するか示したものを**図 6** に示す．まず，製品のテスト中に SBOM-AC が実行されようとするプログラムを監視し，SBOM と不一致なプログラムをログとして全て出力する．開発者は，このログの情報をもとに，SBOM と不一致なプログラムを診断する．診断の結果，不審なプログラムであった場合は適切に対処し，不審なプログラムでなかった場合は SBOM にこのプログラムの情報を含める．

このユースケースの欠点として，悪意のあるプログラムが開発中に実行されない対策がされていた場合は，このプログラムを発見できないというものがある．この場合は，次の節で示す 2 つ目のユースケースで対処する．

4.3 リリース後におけるユースケース

SBOM-AC の機能を有効にした状態で製品をリリースし，リリース後に実行されようとするプログラムを常に監視し制御する．これにより，開発中に SBOM から発見できなかった悪意のあるプログラム，および 3.3 節で示したリリース後に混入した悪意のあるプログラムに対処できる．

このとき，SBOM-AC が行う対処は，ロギングと実行拒否の両方が望ましい．リリース後に不審なプログラムが実行されることはできる限り避けたい．また，実行が拒否されたプログラムを記録し，セキュリティの知識のある人物が適切な対応を実施できる．

このユースケースが脅威をどのように解決するか示したものを**図 7** に示す．まず，SBOM-AC は製品の動作中に実行されようとするプログラムを全て監視する．SBOM と不一致なプログラムを検知した場合，このプログラムの実行を拒否する．また，拒否したプログラムをログに残す．この情報をもとにセキュリティの知識のある人物が拒否されたプログラムに対して適切な対応を行い，問題への対処と再発防止をする．

4.4 制約と許容性の考察

4.4.1 制約と許容性の考察の概要

SBOM-AC の制約事項は，以下の 3 つである．

(1) SBOM に含まれるプログラムの情報に過不足がある場合に対処できない場合がある
(2) 実行される正規ファイルのパスとハッシュ値の情報が SBOM に含まれる必要がある
(3) SBOM-AC そのものを回避された場合に対処できない

以降の項では，それぞれの制約事項について説明し，これが許容できるものか考察する．これにより，SBOM-AC の制約事項は，脅威モデルを解決する上で大きな障害にはならないことを示す．

4.4.2 SBOM のプログラム情報の過不足への対処

SBOM-AC は，SBOM にシステムに存在するプログラムの情報が含まれない（不足する）状況には対処できる．これは，SBOM-AC が SBOM からアクセス制御ポリシーを生成しているため，SBOM に含まれないプログラムは，アクセス制御時のファイル情報の記録や実行拒否のログでこのことを検知でき，対処できるためである．このような状況の発生を防ぐには，SBOM と製品内のシステム全体のプログラムを比較する必要がある．しかし，SBOM はベストエフォートで作成される．このため，SBOM が網羅的に記載されることは求められていない．SBOM-AC では，SBOM に含まれないプログラムを無条件で実行許可することはないため，影響は小さいといえる．

一方，システムに存在しないプログラムの情報が SBOM に含まれる場合，3.2 節で述べた通り，脆弱性管理の問題がある．しかし，これに対して完全には対処できない．これは，SBOM-AC はプログラムが実行されようとしたときのみに検知可能であるためである．SBOM 全体と製品内のプログラムを比較するわけではないため，使われなくなり削除されたコンポーネントの情報が SBOM や SBOM から

生成したアクセス制御ポリシーに残っていても検知できない．SBOM に存在しないプログラム情報が含まれる原因が，SBOM が更新されていないことであった場合，SBOM の作成日を参照することで確認できる．削除したプログラムの SBOM からの削除漏れがあった場合でも，SBOM の情報とコンポーネントのハッシュ値を比較すればよい．

4.4.3 実行される正規ファイルのパスとハッシュ値の情報を含む SBOM の作成

実行される全てのプログラムについて，SBOM にファイルパスとハッシュ値の情報を含める必要があるものの，SBOM にパスとハッシュ値の情報を含めることは，CISA と CRA のどちらも明確に示されていない．また，コンポーネントのハッシュを含めることは CISA [6] やドイツの BSI [9] で言及されているものの，プログラムそのもののハッシュについての言及はない．しかし，今回の脅威に対処するためには，この 2 つの情報は必要不可欠である．

以下, 既存の MAC ポリシーと比較し, SBOM-AC がファイルパスとハッシュ値を用いることの有用性を示す．既存の代表的な MAC システムとして，SELinux と AppArmor を挙げる．SELinux は，ファイルにラベルを付与してポリシーをラベルで記述する．しかし，このポリシーは大規模かつ複雑であり，人間が全容を把握して運用することは困難である．一方，AppArmor は，ポリシーに制御したいファイルのパスを記述する．しかし，ファイルが実際に書き換えられたかどうかを検知する機能は備えていない．

SBOM-AC は，SBOM からファイルパスとハッシュ値を取得することでこの問題を解決している．ポリシーを SBOM から作成することで，人間がポリシーの全容を把握しなくともアクセス制御ができる．また，SBOM に含まれるファイルパスとハッシュ値の組の情報をもとにアクセス制御することで，ファイルが書き換えられた場合も検知と対処ができる．

4.4.4 SBOM-AC そのものを回避されることへの対処

SBOM-AC は，単独ではそれ自身を保護する機能や仕組みを有していない．このため，システムに SBOM-AC が導入されていることを攻撃者が知っていれば，不一致の検知をバイパスできる可能性がある．例えば，SBOM-AC の機能の無効化や細工したポリシの読み込みなどが考えられる．

SBOM-AC を利用する際には，SBOM-AC を保護できる別の仕組みや手法を併用することが望ましい．例えば，SELinux など他の MAC を用いて SBOM-AC に対する操作（ポリシ読み込みなど）を制御することが考えられる．SBOM-AC は他の MAC と競合しないため，このような併用が可能である．

5. 提案手法のプロトタイプの実装と検証

5.1 プロトタイプの実装と検証の概要

SBOM-AC が，SBOM の情報と不一致のプログラムが実行されようとしたときの検知と対処ができるか検証した．具体的には，SBOM-AC のプロトタイプを作成し，以下の 3 つを検証した．

(1) SBOM の内容と不一致のプログラムが実行されようとしたとき，これを検知できるか
(2) 検知したプログラムに対して，ロギングができるか
(3) 検知したプログラムに対して，実行拒否ができるか

なお，検知に成功したかはロギングをしなければわからないため，(1) の検知と (2) のロギングは同時に検証する．

実装および検証に使用した Linux のバージョンは 6.11.11，ディストリビューションは Ubuntu 24.04.2 LTS である．

5.2 提案手法のプロトタイプ実装

SBOM-AC のプロトタイプは，Linux Security Modules（LSM）として MAC を実装する形式で作成した．LSM の仕組みを利用することで，他の MAC と共存する形で，プログラムが実行されようとした際に実行に対する制御を含め独自の処理を行うことができる．処理内容として，実行されようとするプログラムのファイルパスとハッシュ値を求め，それらの情報を LSM に読み込まれたポリシーと照合する．一致しなかった場合にロギングや実行拒否をする．

ポリシーは，別途作成したポリシー作成プログラムを用いて SBOM から作成し，LSM に読み込む．具体的には，SBOM からファイルパスとハッシュ値の組を抽出し，抽出した組のリストを LSM に読み込む．LSM に読み込まれた組のリストは，ハッシュテーブルを用いて連想配列としてメモリ上に保持する．これにより，ファイルパスをキーとしてファイルハッシュ値を参照することができる．

5.3 検証に使う SBOM

5.3.1 検証に使う SBOM の作成

SBOM は，SBOM 作成ツールである Syft のバージョン 1.29.1 を用いて作成した．Syft は，できる限り多くのファイルを収集するよう設定することで，ファイルシステム内を網羅的に探索して見つかった全ファイルのパスとハッシュ値の組を取得可能である．また，SBOM のフォーマットは SPDX のバージョン 2.3 を指定した．SPDX は，一般的な SBOM フォーマットの一つであり，ファイルのパスおよびハッシュ値を記載可能である．

5.3.2 検証に使う SBOM の制約についての議論

今回の検証に用いる SBOM は，ファイルパスのバリエーションに乏しい．このため，どのようなファイルパスでも機能が正しく動作するかの検証はできない．

また，SBOM に含まれるハッシュ値は，プロトタイプ実装の都合により特定のハッシュアルゴリズムのものに固定している．このため，特定のハッシュアルゴリズム以外のハッシュ値でも正しく動作するか，および複数のハッシュアルゴリズムのハッシュ値が混在する SBOM でも正しく

動作するかの検証はできない．

今回の検証は，SBOM の情報と不一致のプログラムの検知ができることを確認できればよいため，上記の制約にかかわらず実施できる．

5.4 提案手法の実現可能性の検証

5.4.1 検証の事前準備

SBOM-AC の検証に使用するために，2 つのファイル（以降，検証ファイル）を作成した．1 つ目は test.bin ファイルで，実行されると test.bin was executed. と標準出力へ出力する．2 つ目は test-new.bin ファイルで，実行されると test.bin ファイルとは異なる出力がされる．この 2 つを/home/user/sbomac_test/直下に配置した．

/home/user/sbomac_test/に SBOM-AC が制御する範囲を限定して実施した．これは，検証ファイル以外のログ出力を抑制し，ログの確認をしやすくするためである．

SBOM は，test.bin ファイルのパスとハッシュ値を含み，test-new.bin の情報を含まないものを作成した．これにより，SBOM-AC は，test.bin のファイル名または内容が変更されて実行されたとき，および test-new.bin ファイルが実行されたときにこれを検知する．

5.4.2 検知とロギングの検証

SBOM-AC がパスとハッシュ値のそれぞれの不一致を検知することができるか検証した．検知された場合，不一致だが実行を許可したことを示すラベル AUDIT と検知したプログラムのファイルパスがログとして出力される．これを確認することで，SBOM-AC が検知とロギングをできるか否かを検証した．なお，この検証は SBOM-AC の実行拒否の機能は無効化した状態で行った．

具体的には，SBOM-AC が実行されようとするプログラムに対して，以下の 3 つの状況のいずれかに当てはまった場合これを検知できるか確認した．

(1) SBOM にファイルパスとハッシュ値のどちらとも一致する情報が記載されていない状況
(2) SBOM にハッシュ値が一致するプログラムの記載はあるがファイルパスが不一致の状況
(3) SBOM にファイルパスが一致するプログラムの記載はあるがハッシュ値が不一致の状況

以上の 3 つを確認することで，SBOM-AC が不一致の検知を行えるか網羅的に検証できる．

それぞれの検知を検証するため，5.4.1 節で作成した検証ファイルを，複製や差し替えを行いつつ実行した．(1) の検知を確認するため，test-new.bin ファイルを実行した．(2) の検知を確認するため，test.bin ファイルの複製として test-copied.bin ファイルを作成し，これを実行した．(3) の検知を確認するため，test.bin ファイルの内容を test-new.bin ファイルに差し替え，これを実行した．

評価の結果をそれぞれ**図 8**，**図 9**，および**図 10** に示す．

```
## SBOM に含まれていないファイルを実行
$ ./test-new.bin
test-new.bin was executed.
## カーネルログに監査ログが出力されている
$ sudo dmesg | grep SBOM-AC
[  134.587413] SBOM-AC: AUDIT /home/user/sbomac_test/test-new.bin
```

図 8 パスとハッシュ値の情報が不一致の場合の検知結果

```
## パス情報が不一致のファイルを実行
$ cp test.bin test-copied.bin
$ ./test-copied.bin
test.bin was executed.
## カーネルログに監査ログが出力されている
$ sudo dmesg | grep SBOM-AC
[  288.658873] SBOM-AC: AUDIT /home/user/sbomac_test/test-copied.bin
```

図 9 パスの情報が不一致の場合の検知結果

```
## ハッシュ値が不一致のファイルを実行
$ cp test-new.bin test.bin
$ ./test.bin
test-new.bin was executed.
## カーネルログに監査ログが出力されている
$ sudo dmesg | grep SBOM-AC
[  321.244954] SBOM-AC: AUDIT /home/user/sbomac_test/test.bin
```

図 10 ハッシュ値の情報が不一致の場合の検知結果

```
## SBOM に含まれていないファイルを実行→拒否される（root にかかわらず）
$ ./test-new.bin
bash: ./test-new.bin: Operation not permitted
$ sudo ./test-new.bin
sudo: unable to execute ./test-new.bin: Operation not permitted
## カーネルログに拒否時のログが出力されている
$ sudo dmesg | grep SBOM-AC
[  430.554603] SBOM-AC: DENIED /home/user/sbomac_test/test-new.bin
[  432.596295] SBOM-AC: DENIED /home/user/sbomac_test/test-new.bin
```

図 11 実行拒否の検証結果

3 つの状況全てにおいて，検知対象ファイルの不一致を検知したログが出力された．これにより，SBOM-AC は不一致の検知とロギングを行えるといえる．

5.4.3 実行拒否の検証

SBOM-AC が不一致を検知した際に実行拒否できるかどうか検証した．SBOM-AC の実行拒否の機能を有効化し，不一致のプログラムが実行されないことを確認した．確認方法は，プログラムが実行されたことを示す文字列が出力されていないかで確認した．なお，このとき出力されるログは，不一致のため実行を拒否したことを示すラベル DENIED と SBOM-AC が検知したプログラムのファイルパスである．test-new.bin ファイルをユーザ権限とルート権限の両方で実行させることにより検証した．

評価の結果を**図 11** に示す．ロギングの検証結果と同じように作成したファイルの不一致を検知したログが出力された．このとき，ユーザー権限とルート権限両方の実行にてプログラムが実行されたことを示す文字列が出力されなかった．これにより，SBOM-AC は SBOM と不一致のプログラムに対して実行拒否を十分にできることが分かる．

6. 関連研究

SBOM の問題点について，いくつか研究がされている．例えば，SBOM 作成ツールの評価 [8], [10] では，これらで作成した SBOM の正確性が低いことを示している．また，SBOM についての実態調査 [11], [12] では，SBOM 普及や運用の課題について示されている．

JBomAudit [13] では，JAR ファイルを対象に，SBOM の依存関係の正確性と完全性を評価するツールが開発されている．これにより，25,882 件の JAR ファイルと SBOM を比較して 7,907 件の SBOM に直接的な依存関係が記されていないことを発見された．

SBOM-AC は，SBOM と実際のシステム構成の不一致に対処する方法である．SBOM そのものではなく，実行されようとするプログラムの監視と対処をする．これにより，SBOM 作成後のシステム構成変更にも対処できるようにしている．

7. おわりに

本稿では，SBOM と実際に動作するプログラムの情報の不一致を MAC により対処する SBOM-AC を提案した．SBOM-AC はリリース前とリリース後の両方で使用され，SBOM に情報が正確に記載されていないプログラムが実行されようとすることの記録や実行拒否をする．実行されようとするプログラムを一意に特定するために扱われる情報として，パスとハッシュ値の組を使用する．これにより，SBOM を用いた脆弱性管理による既存の脆弱性，および SBOM に記載されていない悪意のあるプログラムの見逃しを防ぐ．また，SBOM-AC が実際にロギングと実行拒否ができるか検証した．検証結果は，SBOM-AC は不一致を検知して両方の対処ができることを示した．

今後の課題として，SBOM-AC のオーバヘッドを確認し，これの使用が製品のパフォーマンスに大きな影響を与えないか調査することがある．また，SBOM-AC のフィジビリティを調査するためにベンダインタビューを行うことがある．

謝辞 本研究の一部は，岡山工学振興会特別研究，および JST 国家戦略分野の若手研究者及び博士後期課程学生の育成事業（博士後期課程学生支援）JPMJBS2403 の支援を受けたものです．

参考文献

[1] The Linux Foundation: The State of Software Bill of Materials (SBOM) and Cybersecurity Readiness , The Linux Foundation (online), available from ⟨https://www.linuxfoundation.org/research/the-state-of-software-bill-of-materials-sbom-and-cybersecurity-readiness⟩ (accessed 2025-08-17).

[2] Forescout: Popular OT/IoT Router Firmware Images Contain Outdated Software and Exploitable N-Day Vulnerabilities Affecting the Kernel, Forescout (online), available from ⟨https://www.forescout.com/press-releases/ot-iot-router-firmware-outdated-software-vulnerabilities/⟩ (accessed 2024-05-06).

[3] White House: Improving the Nation's Cybersecurity, White House (online), available from ⟨https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity⟩ (accessed 2025-08-17).

[4] EU: The Cyber Resilience Act, EU (online), available from ⟨https://www.cyberresilienceact.eu/⟩ (accessed 2025-08-17).

[5] 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，経済産業省（オンライン），入手先 ⟨https://www.meti.go.jp/press/2024/08/20240829001/20240829001-1r.pdf⟩（参照 2025-08-21）．

[6] CISA: Framing Software Component Transparency: Establishing a Common Software Bill of Materials (SBOM) Third Edition, CISA (online), available from ⟨https://www.cisa.gov/sites/default/files/2024-10/SBOM Framing Software Component Transparency 2024.pdf⟩ (accessed 2025-08-17).

[7] EU: Annexes to the REGULATION OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL ON HORIZONTAL CYBERSECURITY REQUIREMENTS FOR PRODUCTS WITH DIGITAL ELEMENTS, EU (online), available from ⟨https://www.cyberresilienceact.eu/the-cyber-resilience-act-annex-eu⟩ (accessed 2025-08-17).

[8] Yu, S., Song, W., Hu, X. and Yin, H.: On the Correctness of Metadata-Based SBOM Generation: A Differential Analysis Approach, *2024 54th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 29–36 (online), DOI: 10.1109/DSN58291.2024.00018 (2024).

[9] BSI: Technical Guideline TR-03183: Cyber Resilience Requirements for Manufacturers and Products Part2: Software Bill of Materials (SBOM), BSI (online), available from ⟨https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/TechGuidelines/TR03183/BSI-TR-03183-2-2_0_0.pdf⟩ (accessed 2025-08-17).

[10] Dalia, G., Visaggio, C. A., Di Sorbo, A. and Canfora, G.: SBOM Ouverture: What We Need and What We Have, *Proc. 19th International Conference on Availability, Reliability and Security*, ARES '24, (online), DOI: 10.1145/3664476.3669975 (2024).

[11] Bi, T., Xia, B., Xing, Z., Lu, Q. and Zhu, L.: On the Way to SBOMs: Investigating Design Issues and Solutions in Practice, *ACM Trans. Softw. Eng. Methodol.*, Vol. 33, No. 6 (online), DOI: 10.1145/3654442 (2024).

[12] Stalnaker, T., Wintersgill, N., Chaparro, O. et al.: BOMs Away! Inside the Minds of Stakeholders: A Comprehensive Study of Bills of Materials for Software Systems, *Proc. IEEE/ACM 46th International Conference on Software Engineering*, p. 13 (online), DOI: 10.1145/3597503.3623347 (2024).

[13] Xiao, Y., Kirat, D., Schales, D. L. et al.: JBomAudit: Assessing the Landscape, Compliance, and Security Implications of Java SBOMs, *Network and Distributed System Security Symposium*, (online), DOI: 10.14722/ndss.2025.240322 (2025).