JAIST Repository
https://dspace.jaist.ac.jp/

Title	【課題研究報告書】DevSecOps スキームにおいて脆弱性逓減を可能にする OSS ベースの環境調査
Author(s)	森, 健太郎
Citation	
Issue Date	2023-09
Type	Thesis or Dissertation
Text version	author
URL	http://hdl.handle.net/10119/18754
Rights	
Description	Supervisor: 篠田 陽一, 先端科学技術研究科, 修士(情報科学）

JAPAN ADVANCED INSTITUTE OF SCIENCE AND TECHNOLOGY
Japan Advanced Institute of Science and Technology

課題研究報告書

DevSecOps スキームにおいて脆弱性逓減を可能にする OSS ベースの環境調査

森 健太郎

主指導教員　　篠田　陽一

北陸先端科学技術大学院大学
先端科学技術研究科
（情報科学）

令和 5 年 9 月

Abstract

DevSecOps is a technique for automating the integration of development, security, and operations in software development, to consider security at every stage of the software development life cycle. However, managing vulnerabilities in a DevSecOps environment presents several challenges, including understanding the type and impact of vulnerabilities, speeding up vulnerability detection and resolution, and preventing vulnerabilities from recurring.

In this research, I present an open-source software (OSS)-based methodology to address these challenges and create an environment that enables effective vulnerability management in a DevSecOps scheme. This methodology includes threat analysis technologies such as STRIDE, continuous security testing, automated vulnerability scanning, policy checking, and security auditing, as well as Kubernetes for container orchestration, GitHub for source code scanning, and cloud services for constructing a DevSecOps environment.

I applied this methodology to a DevSecOps environment built on cloud services and investigated its efficacy in managing vulnerabilities. The results showed that the method enables the detection of vulnerabilities in the development and operational phases and demonstrated that the method is suitable for vulnerability management in a DevSecOps environment by improving the understanding of vulnerability types and impacts and responding to them.

目次

第 1 章 はじめに 1
1.1 研究の背景 1
1.2 研究の目的 2
1.3 研究の構成 2
第 2 章 DevSecOps に関する基礎知識 3
2.1 DevOps の概要 3
2.2 DevSecOps の概要 4
2.3 DevSecOps の重要性と課題 5
第 3 章 脆弱性の種類と現状 7
3.1 脆弱性の分類 7
3.2 脆弱性管理の現状と課題 9
3.3 セキュリティ対策の取り組み 11
第 4 章 脆弱性逓減手法 12
4.1 脅威分析手法の導入 12
4.2 継続的なセキュリティテストの導入 14
4.3 自動化された脆弱性スキャンの適用 14
4.4 ポリシーチェックとセキュリティ監査の適用 15

第 5 章 関連技術 17
5.1 Container 17
5.2 Kubernetes 18
5.3 GitHub 20
5.4 Policy Agent 22
5.5 Container Registry 24
第 6 章 DevSecOps 環境の検証 25
6.1 検証環境概要 25
6.2 脆弱性診断の流れ 27
6.3 検証結果 33
第 7 章 考察 34
7.1 構築結果への考察 34
7.2 セキュリティ対策への考察 36
第 8 章 おわりに 38

図目次

1.1：DevOps とは 1
3.2：DevSecOps のライフサイクル 10
4.1：Threat Modeling Tool feature 12
4.2：CI / CD Pipeline.. 16
6.1：OSS ベースの DevSecOps 環境構成図... 29
6.2：Harbor　脆弱性スキャン結果例 30
6.3：Kube-hunter 脆弱性スキャン結果例... 30
6.4：Kube-bench 脆弱性スキャン結果例 31
6.5：OWASP-ZAP 脆弱性スキャンダッシュボード 31
6.6：Grafana ダッシュボード 32
6.7：ELK スタック ダッシュボード... 33

表目次

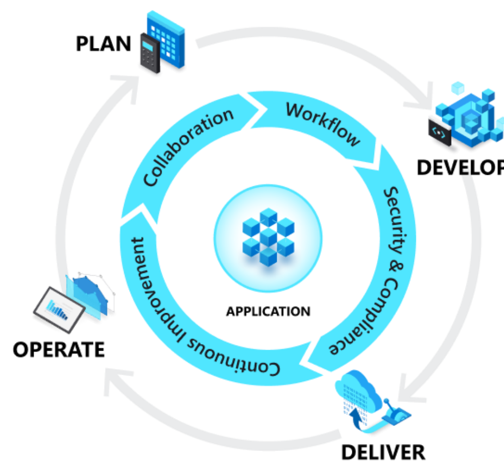
3.1: 表 3.1 SQL Injection Python Code Sample.. 8
3.2: 表 3.2 Buffer Overflow C Code Sample 8
4.1: 表 4.1 STRIDE の 6 つのカテゴリーと定義...................................... 13
5.2: 表 5.2 ReplicaSet 19
6.1: 表 6.1 Open Policy Agent 26
6.2: 表 6.2 Calico Network Policy 26

第1章 はじめに

本章では、DevOps の概要とセキュリティ意識の高まりに伴い変化してきた DevOps の問題点を挙げ、本研究の目的を述べる。

1.1 研究の背景

近年、ソフトウェア開発において、図 1.1 に示す DevOps[1] という開発手法が注目されている。DevOps とは、開発（Development）と運用（Operations）のチームを融合させ、ビジネスにおけるシステムの有用性を高め、迅速な実装と運用を実現する概念である。DevOps は、アジャイル開発［2］とともに、変化の多い開発状況に対応するための効率的な手法として広く普及している。



Source: What is DevOps? [1]
図 1.1 DevOps とは

しかし、DevOps は、セキュリティの脆弱性対策が十分に行われていないという問題が指摘されている。開発期間の短縮と効率化、システムの品質を優先すると、セキュリティの脆弱性対策のための余力がなくなる。また、多くのセキュリティの問題は運用時に明らかになるが、DevOps のライフサイクルで開発時にフィードバックする場合、開発スピードを犠牲にしなければならない。

この課題に対応するために、DevOps から派生して DevSecOps [3]という概念が提唱された。DevSecOps とは、DevOps にセキュリティ（Security）を加え

た概念であり、開発と運用の各フェーズにセキュリティ対策を組み込み、自動化することを目指す。この開発手法では、セキュリティは分離したソリューションではなく、開発チームや運用チームの共通のミッションとして位置付けられる。

1.2 研究の目的

本調査論文は、DevSecOps スキームにおいて脆弱性逓減を可能にする OSS ベースの環境構築について調査する。具体的には、まず DevSecOps スキームにおける脆弱性の種類と現状を分析し、どのようなセキュリティ課題が存在するかを明らかにする。次に、脆弱性逓減手法を紹介し、どのようなセキュリティ対策が有効であるかを整理する。さらに、DevSecOps スキームにおける運用手法を紹介し、どのようなセキュリティ管理が必要であるか明らかにする。最後に、DevSecOps スキームにおいて OSS ベースの環境構築がもたらすメリットや課題を検討し、OSS ベースの環境構築の効果を示す。

1.3 研究の構成

本論文は、以下のように構成されている。第 2 章では、DevSecOps に関する基礎知識を紹介する。第 3 章では、脆弱性の種類と現状を分析する。第 4 章では、脆弱性逓減手法を説明し、その特徴と利点を説明する。第 5 章では、DevSecOps 環境を実装する際に必要になる主要技術に関して説明する。第 6 章では、実環境を実際に構築し、その効果と問題点を検証する。第 7 章では、考察として、検証を通して得られた知見と考察、今後の課題を述べる。第 8 章ではおわりにとして本研究の貢献を述べる。

第2章 DevSecOps に関する基礎知識

2.1 DevOps の概要

本章では DevOps 及び DevOps から発展した概念である DevSecOps に関する概要をまとめ、重要性と課題点を整理する。DevOps とは、開発（Development）と運用（Operations）を組み合わせた造語である。アプリケーションの開発チームと運用チームが協力することにより、迅速かつ柔軟なサービス提供を行うための考え方や仕組みを指す。DevOps では、アジャイル開発の手法を用いることが多く、機能ごとのリリースをスピーディに行う。また、テスト期間を短縮し、早期にリリースすることが可能である。DevOps のメリットとしては、サービスの市場投入までの時間の短縮、市場と競争に適応する能力、システムの安定性と信頼性の向上などが挙げられ、計画、開発、継続的インテグレーション (Continuous Integration) [4] 、デプロイ、運用、継続的なフィードバックのプロセスを定期的に繰り返す。

計画段階では開発するサービスに求められる機能を定義し、タスクや進捗管理の方法について計画を立てる。開発段階では計画に従って、コードの記述など開発工程を進める。継続的インテグレーション（CI）段階では開発したコードのビルドとテストを継続的かつ自動的に実行する。1 日に数回、コードのビルドとテストを行うことで、コードの不具合を早期に発見して修正できる。デプロイ段階では開発工程が一通り終わったら、実際の使用環境で使えるように設定する。

運用段階ではリリース後の運用では、サーバーおよびアプリケーションの監視と、トラブル発生時の対応を実施する。継続的なフィードバック段階では、サービスに関するユーザーからの意見や要望を継続的に受け取り、開発プロセスに反映させる。チャットやメールによる問い合わせのほか、一般公開していれば SNS 上のコメントなども、継続的フィードバックに取り入れられる。

DevOps は、仮想化ツールやインフラ管理ツール、CI/CD [5] ツールなど様々なツールを使用して効率化や自動化を図る。また、タスク管理ツールやコミュニケーションツールなども活用してチーム間の連携や情報共有を促進する。

2.2 DevSecOps の概要

DevSecOps とは、DevOps にセキュリティ（Security）を組み込んだ概念である。アプリケーションのライフサイクル全体にわたってセキュリティを確保することを目指す。DevSecOps では、開発者や運用者だけでなく、セキュリティ担当者もチームに参加し、コードやインフラストラクチャのセキュリティチェックや脆弱性診断などを自動化して行う。DevSecOps のメリットとしては、セキュリティ侵害やデータ漏洩などのリスクの低減、コンプライアンスや規制への対応力、顧客や利害関係者への信頼性の向上などが挙げられる。DevSecOps は、計画、開発、継続的インテグレーション（CI)、デプロイ、運用、継続的フィードバックのプロセスから構成される。

計画段階では開発するサービスに求められる機能を定義し、開発のタスクや進捗管理の方法について計画を立てる。また、セキュリティ要件やポリシーも明確にする。 開発段階では計画に従って、コードの記述など開発工程を進める。コードはセキュリティ規約に従って作成し、静的解析ツールや動的解析ツールなどでセキュリティチェックを行う。 継続的インテグレーション（CI）段階では、開発したコードのビルドとテストを継続的かつ自動的に実行する。1 日に数回、コードのビルドとテストを行うことで、コードの不具合や脆弱性を早期に発見して修正できる。また、セキュリティテストや脆弱性診断も CI プロセスに組み込む。 デプロイ段階では開発工程が一通り終わったら、実際の使用環境で使えるように設定する。デプロイ時にもセキュリティチェックや脆弱性診断を行い、問題があれば修正する。 運用段階ではリリース後の運用、サーバーおよびアプリケーションの監視と、トラブル発生時の対応が重要となる。運用時にもセキュリティチェックや脆弱性診断を定期的に行い、問題があれば修正する。継続的フィードバック段階では、サービスに関するユーザーからの意見や要望を継続的に受け取り、開発プロセスに反映させることである。チャットやメールによる問い合わせのほか、SNS 上のコメントなども、継続的フィードバックに取り入れられる。特にセキュリティに関するフィードバックは重要であり、迅速かつ適切に対応する必要がある。 DevSecOps は、DevOps で使用されるツールに加えて、セキュリティツールも使用して脆弱性診断の実施や効率化・自動化を図る。また、タスク管理ツールやコミュニケーションツールなども活用してチーム間の連携や情報共有を促進する。

2.3 DevSecOps の重要性と課題

DevSecOps は、現代のアプリケーション開発において重要な役割を果たす。アプリケーションは、クラウドやモバイルなどの新しい技術やプラットフォームに対応する必要があるが、それらはセキュリティ上の脅威や複雑さを増加させる。DevSecOps は、これらの課題に対応するために必要なアプローチである。

DevSecOps の重要性としては、以下のような点が挙げられる。まず、セキュリティはアプリケーションの品質に直結する。アプリケーションがセキュリティ侵害やデータ漏洩などの被害に遭うと、顧客や利害関係者の信頼を失うだけでなく、法的な責任や損害賠償などのコストも発生する。また、セキュリティ問題を修正するために開発サイクルが遅れることもある。したがって、セキュリティはアプリケーションの品質に直結する要素であり、DevSecOps ではセキュリティを最優先に考える。また、セキュリティはアプリケーションのライフサイクル全体に関わる。従来の開発手法では、セキュリティは開発後に行われることが多く、開発者や運用者とセキュリティ担当者の間に壁があった。しかし、DevSecOps では、セキュリティはアプリケーションのライフサイクル全体に関わるものとして捉えられる。開発者や運用者もセキュリティ担当者もチームメンバーとして協力し、計画から運用までセキュリティを確保することを目指す。そして、セキュリティはアプリケーションの価値に貢献する。DevSecOps では、セキュリティはコストや制約としてではなく、価値として捉えられる。セキュリティを高めることで、顧客や利害関係者への信頼性や満足度を向上させることができる。また、セキュリティ対応を自動化することで、開発スピードや効率性も向上させることができる。

DevSecOps の課題としては、以下のような点が挙げられる。一つ目はセキュリティカルチャの浸透課題である。DevSecOps では、開発者や運用者もセキュリティに関する知識やスキルを持つ必要がある。しかし、セキュリティは専門的な分野であり、開発者や運用者にとってはカバーが難しい領域である。また、セキュリティ担当者も開発や運用に関する知識やスキルを持つ必要がある。したがって、DevSecOps では、チームメンバー全員がセキュリティカルチャを理解し、共有し、実践することが課題である。二つ目の課題はセキュリティツールの選定と導入である。DevSecOps では、セキュリティツールを使用してコードやインフラストラクチャのセキュリティチェックや脆弱性診断などを自動化する。

セキュリティツールは多種多様であり、それぞれに特徴や利点がある。したがって、DevSecOps では、自分たちのアプリケーションや環境に合ったセキュリティツールを選定し、導入することが課題である。 三つ目の課題はセキュリティポリシー[6] と規制への対応である。DevSecOps では、セキュリティポリシーや規制に従ってアプリケーションを開発し、運用する必要がある。しかし、セキュリティポリシーや規制は変化しやすく、複雑である場合がある。また、異なる国や地域においてもセキュリティポリシーや規制は異なる。したがって、DevSecOps では、セキュリティポリシーや規制への対応を迅速かつ正確に行うことが課題である。

第3章 脆弱性の種類と現状

3.1 脆弱性の分類

脆弱性[7] とは、アプリケーションやシステムに存在するセキュリティ上の欠陥や弱点のことで、攻撃者によって悪用される可能性がある。脆弱性は、様々な要因によって発生することがあるが、一般的には以下のような分類がされる。設計上の脆弱性はアプリケーションやシステムの設計段階で生じる脆弱性で、セキュリティ要件やポリシーの不備、脅威モデリング[8] の分析不足、セキュリティ設計原則の無視などが原因となる。設計上の脆弱性は、開発後に発見されると修正が困難である場合が多く、重大な影響を及ぼすことがある。例えば、認証や暗号化などのセキュリティ機能が不十分であったり、入力値の検証やエラー処理などの例外処理が不適切であったりする場合などが挙げられる。実装上の脆弱性はアプリケーションやシステムの実装段階で生じる脆弱性で、コードの記述やビルドの方法に関する問題が原因となる。実装上の脆弱性は、静的解析ツール[9] や動的解析ツール[10] などで検出できることが多く、修正も比較的容易である。しかし、実装上の脆弱性は数が多く発生することがあり、攻撃者によって組み合わせて利用されることもある。例えば、表 3.1 に示す SQL インジェクション[11] や表 3.2 に示すバッファオーバーフロー[12] などのコードレベルの脆弱性や、未使用コードやデバッグコードなどのビルドレベルの脆弱性などが挙げられる。運用上の脆弱性はアプリケーションやシステムの運用段階で生じる脆弱性で、インフラストラクチャや環境に関する問題が原因となる。運用上の脆弱性は、設定ミスやパッチ適用漏れ、アクセス制御の不備などが原因となる。運用上の脆弱性は、インフラストラクチャ管理ツールやセキュリティ監視ツールなどで検出できることが多く、修正も比較的容易である。しかし、運用上の脆弱性は頻繁に発生することがあり、攻撃者によって即座に利用されることもある。例えば、デフォルトパスワードや不要なサービスなどの設定ミスや、既知の脆弱性を修正するパッチの適用漏れなどが挙げられる。

脆弱性は様々な段階や要因によって発生することがあるが、その影響や対策も異なる。したがって、DevSecOps では、脆弱性の分類を理解し、それぞれに適したセキュリティツールやプロセスを選択、導入することが重要である。

表 3.1: SQL Injection Python Code Sample

```
1. import sqlite3
2.
3. def get_user(user_input):
4.        connection = sqlite3.connect('my_database.db')
5.        cursor = connection. cursor()
6.
7.        # Vulnerable code: inserting user input directly into the query
8.        query = "SELECT * FROM users WHERE name = '{user_input}'"
9.        cursor.execute(query)
10.
11.       result = cursor.fetchall()
12.       return result
```

表 3.2: Buffer Overflow C Code Sample

```
1. #include <string.h>
2.
3. void vulnerable_function(char *input) {
4.        char buffer[128];
5.
6.        strcpy(buffer, input);    // This line can cause a buffer overflow
7. }
8.
9. int main() {
10.       char large input[256];
11.       memset(large_input, 'A', 255);
12.       large_input[255] = '¥0';
13.
14.       vulnerable_function(large_input);
15.
16.       return 0;
17. }
```

3.2 脆弱性管理の現状と課題

DevSecOps では、アプリケーションやシステムに存在する脆弱性を管理することが重要である。脆弱性管理とは、脆弱性を発見し、評価し、対処し、報告し、追跡することである。DevSecOps では、以下のような手順で脆弱性管理を行う。

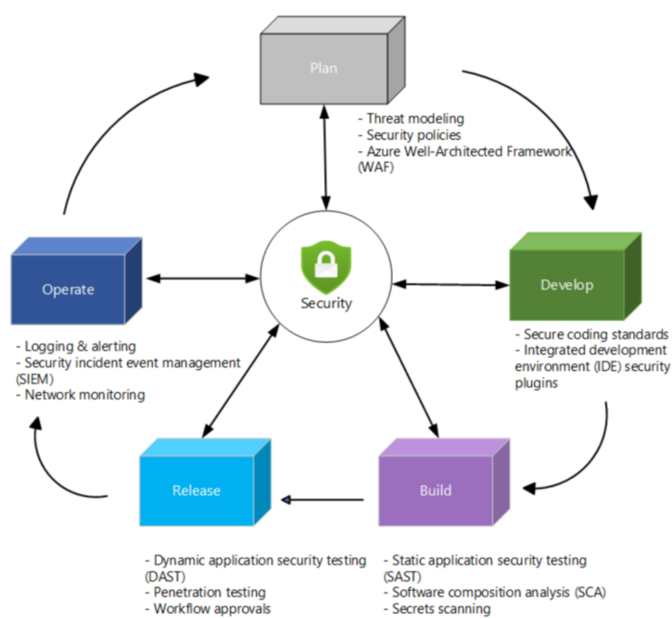
まず計画段階では開発するアプリケーションやシステムに関する情報を収集し、脆弱性管理の目的や範囲、責任者や役割、方法やツールなどを定義する。発見段階ではアプリケーションやシステムに存在する脆弱性を検出する。脅威モデリングやコードレビューなどの手動的な方法や、静的解析ツールや動的解析ツールなどの自動化された方法を使用する。次に評価段階では検出された脆弱性の重要度や優先度を評価する。脆弱性の影響度や発生確率、悪用可能性などを考慮してランク付けする。また、脆弱性の原因や根本原因を分析する。対処段階では評価された脆弱性に対して適切な対策を実施する。対策としては、修正や回避策の適用、リスクの受容などがある。対策の効果を確認するために再テストを行う。最後に報告段階では脆弱性管理の結果や対策の状況を報告する。関係者に対して必要な情報を提供し、説明責任を果たす。追跡段階では脆弱性管理のプロセスや成果物を追跡する。脆弱性管理の履歴や進捗状況を記録し、改善点や問題点を特定する。

DevSecOps での脆弱性管理は図 3.2 に示す継続的かつ自動化されたプロセスとして実施される。[83]　開発者や運用者だけでなく、セキュリティ担当者もチームに参加し、協力して脆弱性管理を行う。また、セキュリティツールやタスク管理ツールやコミュニケーションツールなども活用して、脆弱性管理の効率性を向上させ、早期の対応を可能にする。

脆弱性管理における情報の取り扱いは、その機密性が高いため、特に注意が必要である。情報のアクセス制御と情報共有は、脆弱性管理の重要な側面であり、これらが適切に行われていることが求められる。アクセス制御は、認証と認可のプロセスを通じて、適切なユーザーだけが必要な情報にアクセスできるようにすることを意味する。情報共有は、チームメンバー間で必要な情報が効率的かつ安全に共有されることを保証する。特に新たな脆弱性が発見された場合、その告知範囲や対策がとられるまでの間の情報管理は、DevSecOps のセキュリティ担当者が重点的に注意を払うべき領域である。この期間中、情報の取り扱いは特に

慎重であるべきで、重要度によっては一次対応はチーム内でも一部のメンバーのみに限定するなど適切なアクセス制御と情報共有が行われていることが重要である。一方、情報の暗号化は、情報の保護という観点から重要な要素である。暗号化は情報の配布や共有の方法に関わるものであり、アクセス制御や情報共有とは異なる側面を持つ。暗号化は、情報が不正な第三者によって盗み見られることを防ぐための手段であるが、それ自体が情報の管理や共有を行うものではない。例えば、DevSecOps では、機密性の高い脆弱性情報を共有する際には、SSL/TLS などの暗号化通信プロトコルを用いてデータの安全性を確保する。これにより、情報が転送中に第三者によって傍受されるリスクを軽減する。したがって、暗号化はアクセス制御や情報共有とは異なる観点からリスクを考慮する必要がある。

以上のことを踏まえ、脆弱性管理における現状と課題は、適切なアクセス制御と情報共有の実施、新たな脆弱性の発見とその対策の間の情報管理、そして情報の暗号化という観点から考えることができる。これらの課題に対処することで、DevSecOps における脆弱性管理の効果と効率を向上させることが可能となる。



Source: DevSecOps lifecycle stages [83]
図 3.2 DevSecOps のライフサイクル

3.3 セキュリティ対策の取り組み

DevSecOps においては、アプリケーションやシステムに存在する脆弱性に対して適切なセキュリティ対策を実施することが重要である。セキュリティ対策とは、脆弱性を修正したり回避したりする方法や手段のことで、以下のような種類がある。

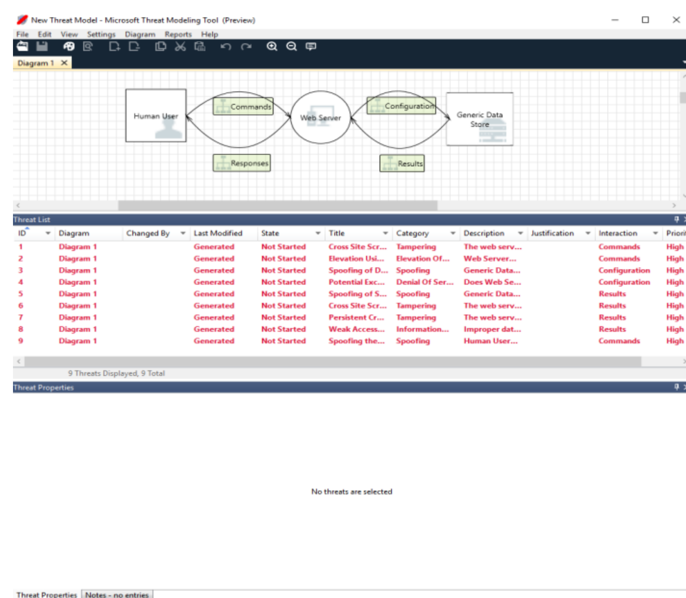
- 修正対策は脆弱性を根本的に解決するために、コードや設定などの変更を行うことである。修正は、脆弱性の再発や悪用を防ぐ最も確実な方法であるが、時間やコストがかかる場合がある。また、修正によって新たな脆弱性や不具合を引き起こす可能性もある。
- 回避対策は脆弱性を直接解決するのではなく、脆弱性の影響や悪用を回避するために、一時的な対処を行うことである。回避策は、修正が困難であったり、緊急であったりする場合に有効であるが、根本的な解決にはならない。また、回避策によってパフォーマンスや機能に影響を与える可能性もある。
- 受容対策は脆弱性を解決するのではなく、脆弱性の存在やリスクを受け入れることである。受容は、脆弱性の影響や確率が低く、修正や回避策のコストが高い場合に有効であるが、悪用される可能性はゼロではない。また、受容には関係者の同意や承認が必要である。

DevSecOps においては、脆弱性管理のプロセスに沿って、各脆弱性に対して最適なセキュリティ対策を選択し、実施する。また、セキュリティ対策の効果や状況を評価し、報告し、追跡する。

第4章 脆弱性逓減手法

4.1 脅威分析手法の導入

脅威分析とは、アプリケーションやシステムに存在する潜在的な脅威やリスクを特定し、評価し、対策するプロセスのことである。脅威分析は、設計段階で行うことが望ましいが、開発や運用の段階でも行うことができる。脅威分析には、DREAD 分析［13］や PASTA 分析［14]など様々な分析手法があるが、ここでは、図 4.1 に示す Microsoft が提唱する STRIDE 分析［15］手法を紹介する。STRIDE とは，Spoofing(なりすまし)，Tampering(改ざん)，Repudiation(否認)，Information disclosure(情報漏洩)，Denial of service, Elevation of privilege(サービス拒否）の頭文字を取ったもので、6 つの主要な脅威カテゴリーを表す[表 1] STRIDE は、以下のような手順で脅威分析を行う。まず、アプリケーションやシステムのアーキテクチャやデータフローを図示する。［16]次にアプリケーションやシステムに関連するアセットやエンティティを特定し、各アセットやエンティティに対して、STRIDE の 6 つのカテゴリーに基づいて、潜在的な脅威を洗い出す。そして、洗い出された脅威に対して、重要度や優先度を評価し、評価された脅威に対して、適切な対策を実施する。



Source: Threat Modeling Tool
図 4.1 Threat Modeling Tool feature [16]

STRIDE の 6 つのカテゴリーとその定義は以下の表 4.1 の通り。

カテゴリ	説明
なりすまし	他のユーザーの認証情報（ユーザー名、パスワードなど）に不正にアクセスし、それを使用する行為。
改ざん	悪意のあるデータの変更など。 例としては、データベースに保持されているような永続的なデータに対する許可されていない変更や、インターネットなどのオープン ネットワーク経由で 2 台のコンピューター間を流れるデータの変更などがある。
否認	反証できる関係者がいない状況でアクションの実行を否定するユーザーに関連するもの。たとえば、禁止されている操作を追跡できる機能がないシステムでユーザーが不正な操作を行うような場合。
情報漏洩	情報へのアクセスが想定されていない個人への情報の暴露など。たとえば、アクセスが許可されていないファイルをユーザーが読み取ることができたり、侵入者が 2 台のコンピューター間で送信されるデータを読み取ることができたりする場合。
サービス拒否	サービス拒否（DoS）攻撃では、有効なユーザーへのサービスが拒否される。たとえば、Web サーバーを一時的に使用できなくする行為。 システムの可用性と信頼性を向上させるために、特定の種類の DoS 脅威からシステムを保護する必要がある。
特権の昇格	特権のないユーザーが特権的なアクセスを取得すると、システム全体を侵害したり破壊したりできるようになる。 特権の昇格の脅威には、攻撃者が効果的にすべてのシステム防御を破り、信頼されているシステム自体の一部となる、本当に危険な状況が含まれる。

Source: STRIDE モデル

表 4.1 STRIDE の 6 つのカテゴリと定義

4.2 継続的なセキュリティテストの導入

セキュリティテストとは、アプリケーションやシステムに存在する脆弱性を検出するために行うテストのことである。セキュリティテストには、静的解析ツールや動的解析 ツールがある。静的解析ツールはアプリケーションやシステムのコードや設定ファイルなどを静的に解析し、脆弱性を検出するツールである。例えば、SonarQube [17] や Veracode[18] などがある。動的解析ツールはアプリケーションやシステムを実行中に解析し、脆弱性を検出するツールである。例えば、OWASP ZAP [19]や Nmap [20]などがある。

4.3 自動化された脆弱性スキャンの適用

脆弱性スキャンとは、アプリケーションやシステムに存在する既知の脆弱性を検出するために行うスキャンのことである。脆弱性スキャンには、以下のような種類がある。ソフトウェアコンポーネント分析(SCA) [21]はアプリケーションやシステムが使用する外部のソフトウェアコンポーネントやライブラリなどに存在する既知の脆弱性を検出するスキャンである。例えば、Black Duck[22]や White Source[23]などがある。コンフィギュレーション管理データベース(CMDB) [24]とはアプリケーションやシステムが使用するインフラストラクチャや環境などに存在する既知の脆弱性を検出するスキャンである。例えば、Qualys [25]や Rapid7 [26]などがあるコンテナイメージスキャン：アプリケーションやシステムが使用するコンテナイメージに存在する既知の脆弱性を検出するスキャンである。例えば、Anchore [27]や Trivy [28] などがある。

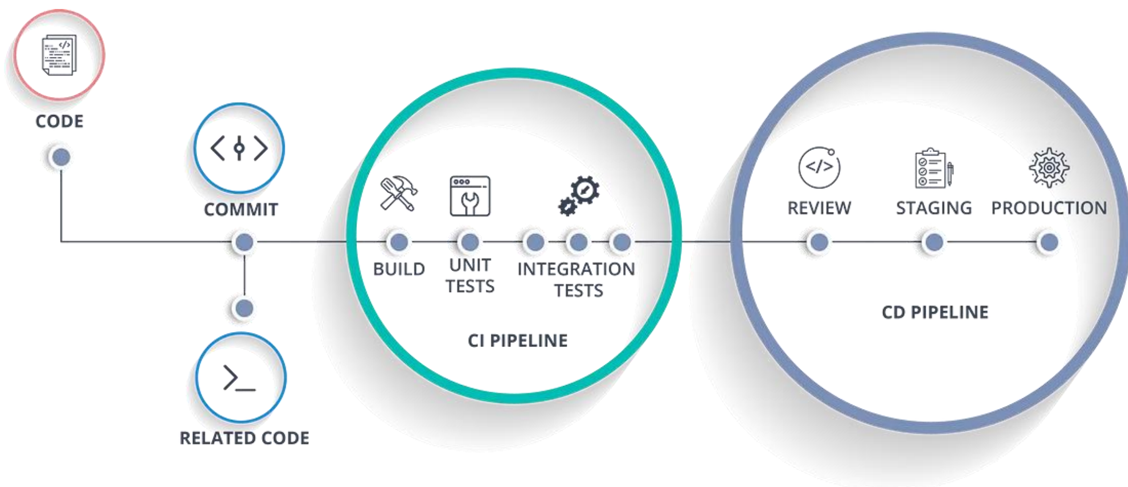
DevSecOps では、セキュリティテストを継続的に行い、開発や運用の各段階で脆弱性を検出し、修正する。また、セキュリティテストを自動化し、CI/CD パイプラインに組み込む。これにより、セキュリティテストの効率や速度が向上し、人為的なミスや遅延が減少する。

4.4 ポリシーチェックとセキュリティ監査の適用

ポリシーチェックとは、アプリケーションやシステムが満たすべきセキュリティ要件や基準に対して、チェックや検証を行うことである。ポリシーチェックには、以下のような種類がある。コード品質チェックとはアプリケーションやシステムのコードが満たすべき品質やセキュリティに関するルールやガイドラインに対して、チェックや検証を行うことである。例えば、SonarQube や ESLint [29] などがある。インフラストラクチャ品質チェックとはアプリケーションやシステムが使用するインフラストラクチャや環境が満たすべき品質やセキュリティに関するルールやガイドラインに対して、チェックや検証を行うことである。例えば、Chef InSpec [30] や Terraform [31] などがある。コンプライアンスチェックとはアプリケーションやシステムが満たすべき法律や規制に関するルールや基準に対して、チェックや検証を行うことである。例えば、PCI DSS [32]や GDPR [33] などの検証項目がある。

セキュリティ監査とは、アプリケーションやシステムのセキュリティ状況やポリシー遵守状況を評価するために行う調査や検査のことである。セキュリティ監査には、以下のような種類がある。内部監査は組織内部のセキュリティ担当者や専門家によって行われる監査である。自主的に行われることが多く、セキュリティ改善のためのフィードバックや勧告を提供する。外部監査は組織外部の第三者機関や専門家によって行われる監査である。法律や規制に基づいて行われることが多く、セキュリティ遵守のための証明や認証を提供する。

DevSecOps では、ポリシーチェックとセキュリティ監査を適用し、アプリケーションやシステムのセキュリティ品質やコンプライアンスを確保する。また、ポリシーチェックとセキュリティ監査を自動化し、図 4.2 に示す CI/CD パイプライン [84]に組み込みを行うことにより、ポリシーチェックとセキュリティ監査の効率や速度が向上し、人為的なミスや遅延が減少する。



Source: What is CI/CD Pipeline? [84]
図 4.2 CI / CD Pipeline

第5章 関連技術

5.1 Container

Linux [34] におけるコンテナ技術は、1979 年に UNIX OS [35] 「Version 7 Unix」の開発途中で生まれた「chroot」というシステムコール／コマンドから始まる。chroot は、アプリケーションのファイルアクセスを特定のディレクトリ以下に限定することで、プロセスを分離するメカニズムである。これにより、システムのセキュリティを向上させることができた。その後、2000 年代に入ると、Linux カーネルに様々な機能が追加された。例えば、cgroups は、プロセスに CPU やメモリなどのリソースを割り当てることができるようにし、また、namespaces は、プロセスに独自のネットワークやユーザーID などの名前空間を与えることができるようにした。これらの機能により、プロセスの分離と制限がより強力になった。Linux Containers プロジェクト (LXC) [36] は、2008 年に登場したオープンソースのコンテナ・プラットフォームで、これらのカーネル機能を利用して、軽量な仮想化環境を提供した。LXC は、シンプルなコマンドライン・インタフェースを備えており、コンテナの作成や管理を容易にした。2013 年には、Docker [37] が登場した。Docker は、LXC をベースにしていたが、コンテナイメージの作成や配布を簡単にするレイヤー構造やレジストリなどの機能を追加した。また、アプリケーションごとにコンテナを分離するマイクロサービスアーキテクチャを推進した。

コンテナ技術は従来の仮想化技術とはアーキテクチャが異なる。仮想化技術では、ハイパーバイザーと呼ばれるソフトウェアを使用して、物理的なハードウェアをエミュレートする。これにより、複数のオペレーティングシステム (Windows や Linux など) を単一のシステム上で同時に実行できる。しかし、この方法はコンテナ技術よりも重量級であり、オーバーヘッドが大きくなる。コンテナ技術では、単一のオペレーティングシステムでネイティブに実行されるが、カーネル機能を利用してプロセスを分離する。すべてのコンテナは同じカーネルを共有するが、それぞれ独自のファイルシステムや名前空間やリソース制限を持つ。この方法は軽量であり、高密度でデプロイすることを可能にする。

5.2 Kubernetes

Kubernetes [38] とは、コンテナ化されたアプリケーションのデプロイやスケーリングを自動化したり、管理したりするためのオープンソースのシステムである。Kubernetes は、Google が 2003 年に導入したコンテナクラスタ管理システム Borg [39] の経験を基にして、2014 年に発表された。Borg では、Linux カーネルの分離メカニズムを利用して、複数のコンテナを効率的に管理していた。Kubernetes は、Borg の概念を引き継ぎながらも、オープンソースであり、様々なコンテナランタイム（Docker など）に対応し、また、クラウドネイティブ・コンピューティング財団（CNCF [40] ）によって管理され、多くの企業やコミュニティからの貢献を受けている。

Kubernetes の中心的な機能は、「コンテナオーケストレーション」である。これは、複数のコンテナを単一のユニットとして機能するように接続し、可用性やスケーラビリティを自動的に管理することである。Kubernetes では、以下のような概念や仕組みを用いてコンテナオーケストレーションを実現している。

- Cluster：単一のユニットとして機能するように接続された、可用性の高いコンピューターの集合。クラスターは、マスターと呼ばれる制御プレーンと、ワーカーと呼ばれるデータプレーンから構成される。
- Pod：Kubernetes が管理できる最小単位で、1 つ以上のコンテナが含まれる。ポッドは、IP アドレスやストレージなどのリソースを共有する。
- Service：ポッドに対する抽象化であり、ポッドへの安定的なアクセス方法を提供する。サービスは、ロードバランシングやサービスディスカバリなどの機能を持つ。
- Deployment：ポッドやレプリカセット（ポッドの複製）の状態を定義し、宣言的に管理する。デプロイメントは、ローリングアップデートやロールバックなどの機能を持つ。
- Namespace：同一の物理クラスター上で動作する複数の仮想クラスターである。ネームスペースは、クラスター内のリソースや名前を分離し、複数のユーザーやチームが共存できるようにする。
- ReplicaSet：宣言した Pod 数を維持するためのワークロードリソースである。表 5.2 の例では、任意のコンテナを使用し、2 つの my-container ポッドを作成し、それらを管理している。

表 5.2: ReplicaSet

```
1. apiVersion: apps/v1
2. kind: ReplicaSet
3. metadata:
4.    name: my-replicaset
5. spec:
6.    replicas: 2
7.    selector:
8.       matchLabels:
9.          app: my-app
10.   template:
11.      metadata:
12.         labels:
13.            app: my-app
14.      spec:
15.         containers:
16.         - name: my-container
17.            image: nginx
18.            ports:
19.            - containerPort: 80
```

5.3 GitHub

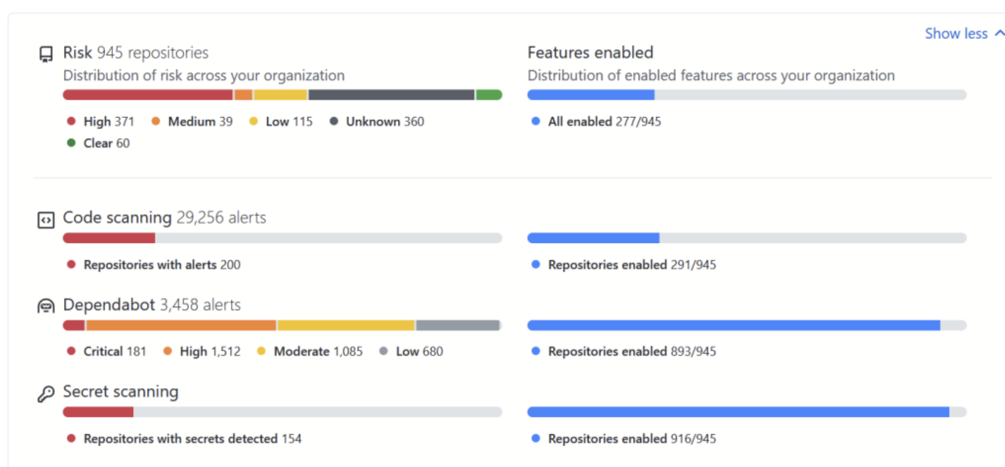
GitHub [41] は、Git [42] という分散型バージョン管理システムを基盤としたインターネットホスティングサービスである。Git は、データの完全性を保証し、高速なソフトウェアビルドを可能にするオープンソースのコード管理システムであり、Linux の開発者である Linus Torvalds [43] によって作成された。GitHub は、Git の分散型バージョン管理機能に加えて、アクセス制御、バグ追跡、機能リクエスト、タスク管理、継続的インテグレーション、プロジェクトごとのウィキなどを提供する。GitHub は 2007 年に Logical Awesome という名前で設立され、2008 年にウェブサイトが公開された。GitHub はオープンソースソフトウェア開発プロジェクトをホストすることで広く知られており、2023 年 1 月時点で 1 億人以上の開発者と 3 億 7 千万以上のリポジトリを有している。GitHub には CI/CD を実現するための GitHub Actions [44] やセキュリティ機能を提供する GitHub Advanced Security [45] などの機能が存在する。

GitHub Actions は、GitHub 上でソフトウェア開発ワークフローを自動化、カスタマイズ、実行できる機能である。GitHub Actions は、CI/CD をはじめとする任意のジョブを実行できるアクションを発見、作成、共有できる。また、アクションを組み合わせて完全にカスタマイズされたワークフローを作成できる。GitHub Actions は、Linux、Mac [46] 、Windows [47] 、ARM [48] 、コンテナなどのさまざまな OS のホストランナーを提供し、VM [49] やコンテナの中で直接実行できる。また、自前の VM やクラウドやオンプレミスの環境にある自己ホストランナーも使用できる。GitHub Actions は、Node.js [50]、Python [51] 、Java [52] 、Ruby [53] 、PHP [54] 、Go [55] 、Rust [56] 、.NET [57] などのさまざまな言語に対応している。

図 5.3 に示す GitHub Advanced Security は、Advanced Security ライセンスの下で利用できる追加のセキュリティ機能である。これらの機能は、GitHub.com 上のパブリックリポジトリでも有効になっている。GitHub Advanced Security は、GitHub Enterprise Cloud [58] のエンタープライズアカウントで利用できる。GitHub Advanced Security には以下の機能がある。

- Code scanning: コードに潜在的なセキュリティ脆弱性やコーディングエラーを検索する。

- Secret scanning: プライベートリポジトリにチェックインされたキーやトークンなどのシークレットを検出する。
- Dependency review: 依存関係に対する変更の影響を全体的に表示し、脆弱なバージョンの詳細をプルリクエストのマージ前に確認する。
- Dashboard: 管理者はセキュリティ設定を有効にした各リポジトリのアクティブなアラートの数と重大度に基づいて分析情報を得る。



Source: GitHub Advanced Security Blog [59]
図 5.3 GitHub Advanced Security Dashboard

5.4 Policy Agent

Policy Agent 技術とは、Kubernetes クラスタに対してポリシーを定義し、実行し、制御できるツールの総称である。Policy Agent 技術の主な実装として、Open Policy Agent (OPA) [60]と Kyverno [61] がある。OPA は、さまざまなプラットフォームに対応した汎用的なポリシーエンジンであり、Rego [62] という専用の言語でポリシーを記述できる。Kyverno は、Kubernetes に特化したポリシーエンジンであり、YAML でポリシーを記述できる。

Policy Agent は、Pod Security Policy (PSP) [63] という Kubernetes のネイティブ機能を用いることが推奨であったが Kubernetes Version 1.21 から非推奨となり、1. 25 から廃止されることになった。PSP は、Pod が実行される前にセキュリティ上の制約を適用するものであったが、以下のような問題があった。

- ポリシーの定義や適用が複雑で優先順位が不明確。
- ポリシーの変更や削除が容易。
- ポリシーの違反時に有効なフィードバックが得られない。

これらの問題を解決するために、OPA や Kyverno などの外部ツールが開発された。これらのツールは、PSP よりも柔軟性や拡張性が高く、ポリシーをコードとして管理しやすくした。これら後続の開発を受け、PSP は 2021 年 11 月に廃止されることが決定された。

PSP の廃止後は、Pod Admission Policy [64] という新しい Kubernetes のネイティブ機能が導入される予定である。Pod Admission Policy は、OPA や Kyverno と互換性があり、以下のような特徴を持つ。

- ポリシーはカスタムリソースとして定義される。
- ポリシーは名前空間やクラスタレベルで適用される。
- ポリシーは優先順位や競合解決ルールに従って評価される。
- ポリシーはドライランモードや監査モードでテストできる。

Policy Agent 技術の主な機能は、アドミッションコントロールと監査である。アドミッションコントロールでは、オブジェクトが作成、更新、削除される際にポリシーに基づいて許可や拒否を行う。例えば、以下のようなポリシーを実施できる。

- すべてのリソースに特定のラベルを要求する。
- コンテナイメージが信頼できるレジストリから来ていることを要求する。

- すべての Pod がリソース要求と制限を指定していることを要求する。
- 矛盾する Ingress オブジェクトの作成を防止する。

監査では、クラスター内のオブジェクトがポリシーに準拠しているかどうかを定期的にチェックし、違反した場合にアラートやレポートを生成する。例えば、以下のような監査が可能である。

- シークレットやキーなどの機密情報がプライベートリポジトリにチェックインされていないか検出する。
- 依存関係に対する変更の影響を全体的に表示し、脆弱なバージョンの詳細を確認する。
- プルリクエストのマージ前にコードスキャンを実行し、潜在的なセキュリティ脆弱性やコーディングエラーを検出する。

5.5 Container Registry

Container Registry 技術とは、コンテナイメージや関連するアーティファクトをプライベートなレジストリにビルド、保存、管理、配信できるツールの総称である。Container Registry 技術は、コンテナの開発とデプロイメントのパイプラインに統合され、コンテナのライフサイクルを簡素化する。Container Registry 技術の主な実装として、Docker Registry [65] などがある。また、OSS の Container Registry として、Harbor [66]や Quay [67] などがある。

Container Registry 技術は、Docker Registry という Docker のネイティブ機能から始まった。Docker Registry は、Docker イメージを保存し、Docker Hub [68] や他のレジストリにプッシュやプルを行うことができる。Docker Registry は、2013 年にリリースされた。Docker Registry の問題点は、セキュリティやスケーラビリティが低く、コンテナイメージ以外のアーティファクトに対応していなかったことである。これらの問題を解決するために、Azure Container Registry や Harbor などの外部ツールが開発された。これらのツールは、Docker Registry よりも高度な機能を提供し、以下のような特徴を持つ。

- コンテナイメージだけでなく、Helm チャート [69] や OCI アーティファクト[70] なども保存できる。
- クラウドサービスやオンプレミス環境に対応し、地域間でレジストリを同期できる。
- Azure Active Directory [71] や Role-based Access Control [72] などの認証・認可機能を備える。
- Docker Content Trust [73] や Notary [74] などの署名・検証機能を備える。

Container Registry 技術の主な機能は、ビルド、保存、配信である。ビルドでは、コンテナイメージをソースコードから作成し、レジストリにプッシュする。保存では、コンテナイメージや関連するアーティファクトをレジストリに格納し、バージョン管理やラベル付けを行う。配信では、コンテナイメージや関連するアーティファクトをレジストリからプルし、コンテナランタイムに渡す。

第6章 DevSecOps 環境の構築

6.1 検証環境概要

本章では、Microsoft のパブリッククラウドサービス Azure 上に DevSecOps の検証環境を構築し、5 段階のセキュリティチェックを実施することで脆弱性の発生数と種類を確認する。この検証環境では、デプロイ前とデプロイ後の各工程にセキュリティチェックを組み込むことで、アプリケーションに脆弱性が入り込むリスクを可視化し、逓減することを目的とする。

利用するコンポーネントは OSS(Open Source Software)ベースで実装する。具体的には、マネージド Kubernetes サービスである Kubernetes Cluster を利用し、コンテナ化されたアプリケーションのデプロイメントと管理を行う。また、GitHub の機能の一部である GitHub Actions を利用して、CI/CD パイプラインを実現する。さらに、オープンソースのコード品質管理プラットフォームである Kube-bench を利用して、静的アプリケーションセキュリティテスト（SAST）を行う。同様に、オープンソースの Web アプリケーションスキャナーである OWASP ZAP を利用して、動的アプリケーションセキュリティテスト（DAST）を行う。Kubernetes 上に展開するコンテナイメージはコンテナレジストリの Harbor からプッシュする。Harbor は Trivy と組み合わせることでイメージレベルの脆弱性検知を可能にするまた、Kubernetes 上のログ・メトリクスを監視機能である Prometheus に集約し、ログフォワーダーの fluentd を中継させ ELK Stack に集約することで、リアルタイムで脅威を自動検出、通知する SIEM (Security Information and Event Management) を実現する。Harbor からデプロイされるコンテナイメージはデプロイ時に Open Policy Agent のポリシーチェックを受ける。表 6.1 のポリシー例では Pod のコンテナイメージが "harbor.io/"で始まらない場合にデプロイを拒否する。またデプロイ後は Calico のネットワークポリシーチェックを受けることで設定したポリシーに反したイメージはデプロイされなくなる。表 6.2 のポリシー例では 192.168.0.0/16 からの入力トラフィックと、10.0.0.0/16 への出力トラフィックを'app'ラベルが'myapp'の Pod に対して拒否するものである。

表 6.1: Open Policy Agent

```
1. package kubernetes.admission
2.
3. deny[msg] {
4.     input.request.kind.kind == "Pod"
5.     image := input.request.object.spec.containers[_].image
6.     not startswith(image, "harbor.io/")
7.     msg := sprintf("invalid image registry: %v", [image])
8. }
```

表 6.2: Calico Network Policy

```
1. apiVersion: projectcalico.org/v3
2. kind: NetworkPolicy
3. metadata:
4.     name: deny-traffic
5.     namespace: default
6. spec:
7.     selector: app == 'myapp'
8.     types:
9.     - Ingress
10.    - Egress
11.    ingress:
12.    - action: Deny
13.        source:
14.            nets:
15.            - 192.168.0.0/16
16.    egress:
17.    - action: Deny
18.        destination:
19.            nets:
20.            - 10.0.0.0/16
```

6.2 脆弱性診断の流れ

今回の検証環境では、まず開発者が GitHub のレポジトリにソースコードを push もしくは既存のレポジトリへ pull request を出した際、GitHub Advanced Security のソースコード診断機能で脆弱性チェックを行う。(図 6.1 ①ソースコードスキャン）この機能は、開発者がコードを書く際のセキュリティ上の問題を早期に発見し、修正することを可能にする。GitHub Advanced Security のソースコード診断機能は、以下のよう脆弱性をチェックすることができる。まず、セキュリティ脆弱性診断はソースコード中の潜在的なセキュリティリスクを検出する。これには、SQL インジェクション、クロスサイトスクリプティング(XSS)、クロスサイトリクエストフォージェリ（CSRF）などの一般的なウェブアプリケーションの脆弱性が含まれる。次にコード品質問題診断はソースコード中の一般的なコーディングエラーやベストプラクティスへの違反を検出する。これには、未使用の変数、未定義の変数、未使用のインポートなどが含まれる。最後に依存関係の脆弱性診断はプロジェクトが依存しているパッケージやライブラリに含まれる既知の脆弱性を検出する。これらの脆弱性が検出されると、GitHub Advanced Security は詳細な情報と修正の提案を提供する。これにより、開発者は脆弱性を早期に検出し、修正することができる。

ソースコード診断後、GitHub Actions の CI 機能でソースコードがコンテナイメージに固められ Harbor レポジトリに格納される。Harbor レジストリに格納されたイメージは図 6.2 に示す Trivy によるイメージスキャン [76]を実施することでイメージレベルの脆弱性診断を実施する。（図 6.1 ②イメージスキャン）Trivy によるスキャンでは主要なプログラミング言語のライブラリに存在する脆弱性を検出することができ、また継続的に脆弱性データベースが更新されることで最新の脆弱性情報に基づいてスキャンを行うことができる。

一連のチェックが終わったイメージが Kubernetes 上にデプロイされる際、Open Policy Agent (OPA)のデプロイポリシーチェックと Calico のネットワークポリシーチェックを通し、認可されていないレポジトリからのデプロイや不必要な Ingress, Egress の通信を制限する。(図 6.1 ③ポリシーチェック）OPA は、デプロイメントのポリシーチェックを行うことで、不適切な設定や権限の過剰な付与などの脆弱性を検出する。また、組織が定めたポリシーに違反するデプ

ロイメントを検出し、組織のセキュリティポリシーを遵守することができる。Calico は Kubernetes のネットワークトラフィックをチェックし、不正なトラフィックや攻撃を検出する。また、組織が定めたネットワークポリシーに違反するネットワークトラフィックを検出し、制限することができる。

Kubernetes のクラスターのセキュリティチェックを実施するため Kube-bench, Kuber-hunter による診断[77]を実施する。(図 6.1 ④クラスタースキャン) 図 6.3 に示す Kube-hunter は、Kubernetes クラスターに対する潜在的な攻撃ベクトルを探すためのツールである。Kube-hunter は、クラスター内外からのアクセスを模擬し、潜在的な脆弱性を検出する。具体的には、公開されている API エンドポイント、不適切な設定による情報漏洩、認証や認可の問題などをチェックする。一方、図 6.4 に示す Kube-bench は、Kubernetes のベストプラクティスをチェックするツールであり、CIS Kubernetes Benchmark に基づいた診断[78]を行う。これにより、Kubernetes の設定に関する脆弱性を検出することが可能となる。具体的には、Kubernetes の各コンポーネント（etcd、kubelet、kube-apiserver、kube-controller-manager など）の設定がセキュリティ上のベストプラクティスに従っているかをチェックする。これにより、不適切な設定による潜在的なセキュリティリスクを早期に検出し、修正することが可能となる。最後に図 6.5 に示す OWASP ZAP を使った定期的なペネトレーションテスト[79]を実施し外部からの攻撃リスクを低減する。(図 6.1 ⑤ペネトレーションテスト) ZAP はリフレクティブ、ストアド、および DOM ベースの XSS 攻撃の検出、SQL インジェクション攻撃の検出、セッション管理の問題の検出など様々な脆弱性診断が可能になる。

Habor レポジトリ内の脆弱性チェック、OPA や Calico の診断で検出されたポリシー違反を含む時系列データは Prometheus で収集され、アラートルールを設定することで、特定の条件が満たされたときに通知を送ることができる。これにより、問題が発生した際に迅速に対応することが可能になる。また、Prometheus から収集されたメトリクスを可視化するため、図 6.6 に示す Grafana のダッシュボード機能 [80]を利用する。これにより、システムの状態を一目で把握することが可能になる。Kube-bench, Kube-hunter, OWASP ZAP で実施した脆弱性診断結果は Fluentd によりログフォワーディングを実施し、宛先の ELK Stack に送付される。図 6.7 に示す ELK Stack は、Elasticsearch、Logstash、Kibana [81]の 3 つのオープンソースソフトウェアからなるログ管理

プラットフォームであり今回の構成ではセキュリティ管理チームが発見された脆弱性のトリアージを実施するために利用される。

また、Vault は、秘密情報、データ、システムへのアクセスを自動化し、保護するために利用する。Vault は、クライアント（ユーザー、マシン、アプリ）を検証および承認し、秘密情報または保管された機密データへのアクセスを提供する前に、認証を行う。Vault は、秘密管理、Kubernetes シークレット、データベース資格情報のローテーション、自動化されたインフラストラクチャ、データの暗号化とトークン化、キー管理などの用途に使用できる。

この図 6.1 の構成[75][85]でシステムを運用する場合、大きく 5 つの段階で脆弱性診断を実施することになるが、全てを開発者が実施するのは作業工数の観点で現実的ではないと考える。そのため開発環境の対応は開発者が、運用環境の対応はセキュリティ担当者や運用担当者が実施するなど組織内での役割分担が必要になってくる。また、脆弱性の種類によっては秘匿性が求められる場合もあるため、一部の脆弱性診断結果は直接セキュリティ管理部門のログデータベースに転送されることがガバナンスの観点から重要であると考える。

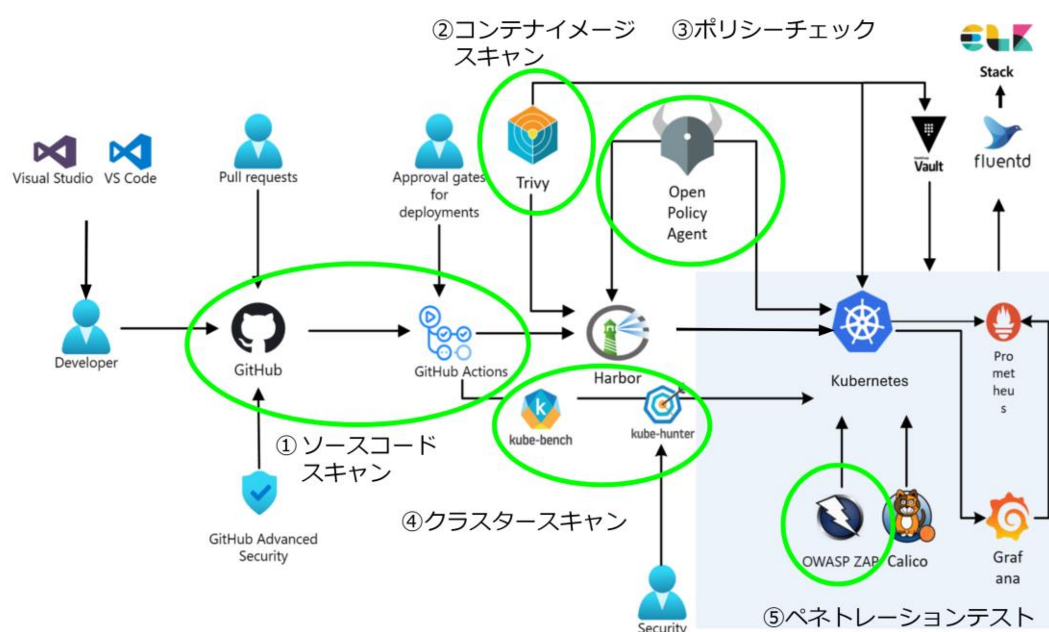
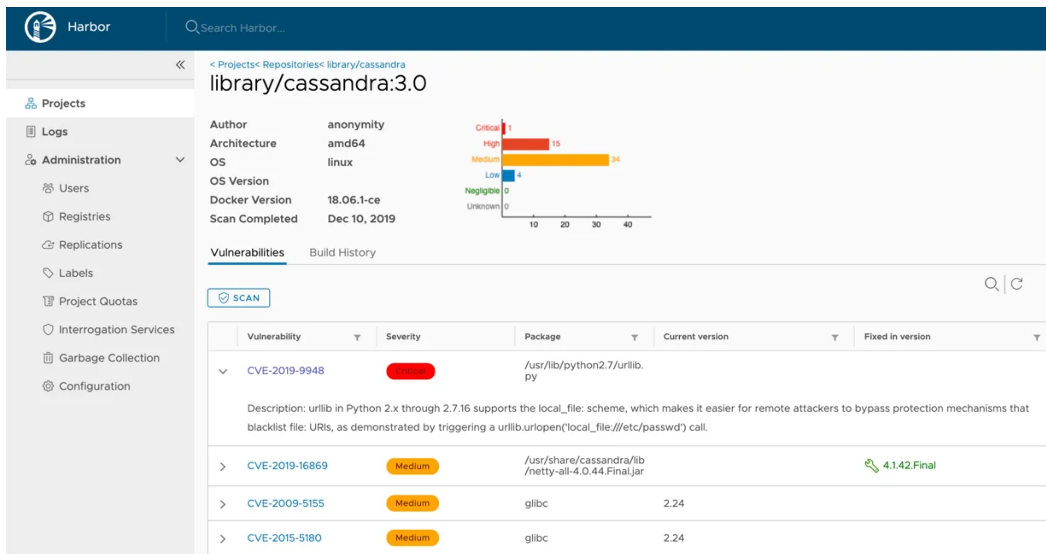
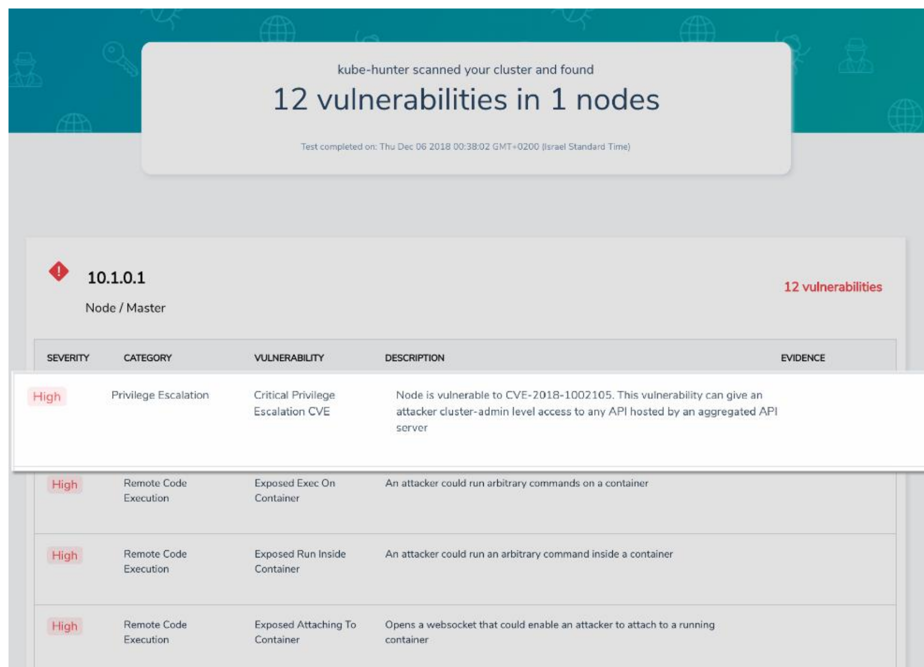


図 6.1 OSS ベースの DevSecOps 環境構成図[75]



Source: Pluggable Image Vulnerability Scanners for Harbor [76]
図 6.2 Harbor　脆弱性スキャン結果例

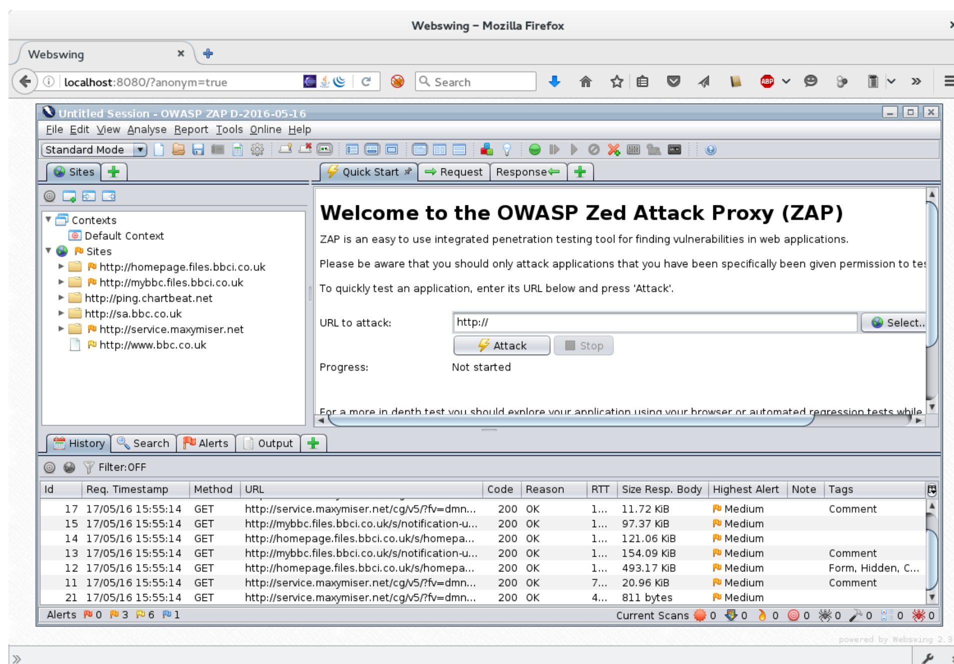


Source: Severe Privilege Escalation Vulnerability in Kubernetes [77]
図 6.3 Kube-hunter 脆弱性スキャン結果例

```
[INFO] 1 Master Node Security Configuration
[INFO] 1.1 API Server
[FAIL] 1.1.1 Ensure that the --allow-privileged argument is set to false (Scored)
[FAIL] 1.1.2 Ensure that the --anonymous-auth argument is set to false (Scored)
[PASS] 1.1.3 Ensure that the --basic-auth-file argument is not set (Scored)
[PASS] 1.1.4 Ensure that the --insecure-allow-any-token argument is not set (Scored)
[FAIL] 1.1.5 Ensure that the --kubelet-https argument is set to true (Scored)
[PASS] 1.1.6 Ensure that the --insecure-bind-address argument is not set (Scored)
[PASS] 1.1.7 Ensure that the --insecure-port argument is set to 0 (Scored)
[PASS] 1.1.8 Ensure that the --secure-port argument is not set to 0 (Scored)
[FAIL] 1.1.9 Ensure that the --profiling argument is set to false (Scored)
[FAIL] 1.1.10 Ensure that the --repair-malformed-updates argument is set to false (Scored)
[PASS] 1.1.11 Ensure that the admission control policy is not set to AlwaysAdmit (Scored)
[FAIL] 1.1.12 Ensure that the admission control policy is set to AlwaysPullImages (Scored)
[FAIL] 1.1.13 Ensure that the admission control policy is set to DenyEscalatingExec (Scored)
[FAIL] 1.1.14 Ensure that the admission control policy is set to SecurityContextDeny (Scored)
[PASS] 1.1.15 Ensure that the admission control policy is set to NamespaceLifecycle (Scored)
[FAIL] 1.1.16 Ensure that the --audit-log-path argument is set as appropriate (Scored)
[FAIL] 1.1.17 Ensure that the --audit-log-maxage argument is set to 30 or as appropriate (Scored)
[FAIL] 1.1.18 Ensure that the --audit-log-maxbackup argument is set to 10 or as appropriate (Scored)
[FAIL] 1.1.19 Ensure that the --audit-log-maxsize argument is set to 100 or as appropriate (Scored)
[PASS] 1.1.20 Ensure that the --authorization-mode argument is not set to AlwaysAllow (Scored)
[PASS] 1.1.21 Ensure that the --token-auth-file parameter is not set (Scored)
[FAIL] 1.1.22 Ensure that the --kubelet-certificate-authority argument is set as appropriate (Scored)
```

Source: Kube-Bench: An Open-Source Tool for Running Kubernetes CIS Benchmark Tests [78]

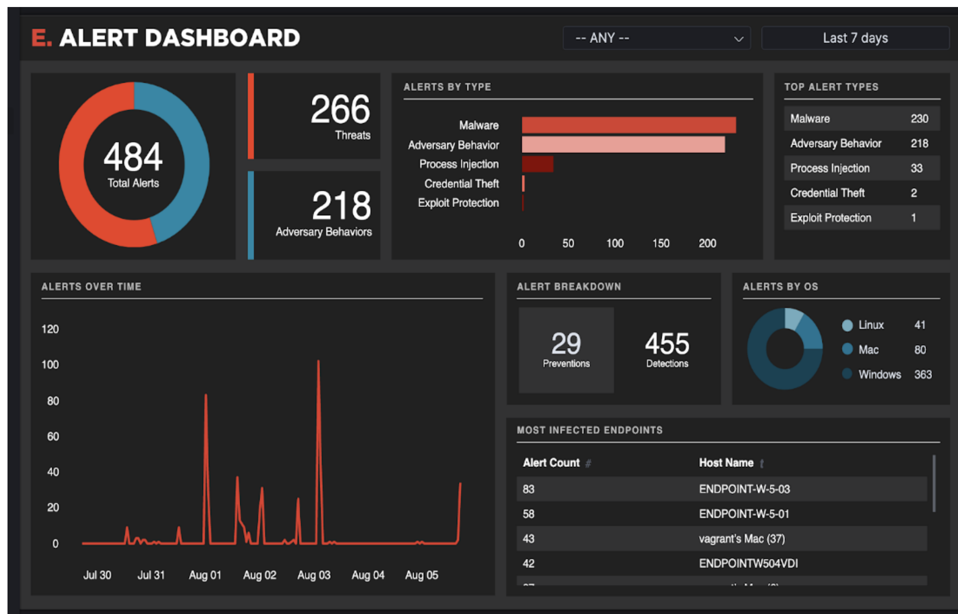
図 6.4 Kube-bench 脆弱性スキャン結果例



Source: ZAP - Webswing Usage [79]

図 6.5 OWASP-ZAP 脆弱性スキャンダッシュボード

Source: GRAFANA SUPPORT FOR PROMETHEUS [80]
図 6.6 Grafana ダッシュボード



Source: Visualizing security data with Canvas [81]
図 6.7 ELK スタック ダッシュボード

6.3 検証結果

この環境を構築し、サンプルアプリ[82] をデプロイした結果、GitHub Advanced Security のソースコード診断機能では脆弱性は発見されなかったが依存関係診断では Node.js パッケージの脆弱性が指摘され、バージョンアップが必要とされた。次にイメージからは約 400 種類の脆弱性が発見された。脆弱性にはホストイメージへのセキュリティパッチが未対応なものや使用している言語のセキュリティアップデートが必要なものが含まれていた。これらの脆弱性はシステム開発チームで対応が必要になる。OPA では Harbor レジストリからのイメージデプロイのみ許可し、Calico ではサンプルアプリの利用する IP セグメントのみ許可することでデプロイを可能にした。Kube-bench の診断では注意レベルの評価のみで修正が必要な脆弱性は発見されなかったが、 Kube-hunter の診断では Pod の露出や Container の露出など約 10 種類の脆弱性が発見された。最後の OWASP ZAP の脆弱性診断では 30 種類ほどの脆弱性が発見された。項目にはセッション管理の問題やセキュリティヘッダーの問題などが検出された。これらの脆弱性はシステム開発チームだけではなくインフラ運用チームでの対応が必要になると考える。OWASP ZAP で発見された脆弱性にはソースコードスキャンやイメージスキャン、ポリシー定義時に発見される脆弱性と重複するものもあった。

第7章 考察

7.1 構築結果への考察

DevSecOps 環境構築の結果から得られた知見及び考察を以下にまとめる。

- DevSecOps は環境構築自体に時間を要する。本研究では、DevOps 開発経験がある場合でも、今回の検証環境の構築に 40 時間ほどかかかった。実運用時には、組織のセキュリティポリシーに対応させる必要があり、これ以上の工数コストがかかると予想される。AWS や Azure などパブリッククラウドベンダーが提供するエンタープライズ製品を中心に環境を構築する場合は、既存のコードテンプレートや開発ドキュメントが豊富なため構築工数を OSS ベースの環境構築より削減できる可能性があるが、ベンダーロックインなどエンタープライズ環境特有の制約を考慮する必要がある。

- Kubernetes 環境にサンプルアプリケーションをデプロイすると、セキュリティポリシーに違反する場合にデプロイが拒否されることがある。例えば、本検証では、Kubernetes クラスターへアプリケーションをデプロイする際に、Open Policy Agent の latest タグの使用禁止などのポリシーからデプロイを拒否された。これはポリシーとして正しい挙動だが、拒否ポリシーに準拠したアプリケーションの修正工数が必要になる。セキュリティポリシーの設定や適用は、アプリケーションのセキュリティ要件やリスク評価に基づいて行われる。ポリシーの内容や厳格さは、アプリケーションの種類や目的、リスクレベルなどに応じて変化する。例えば、個人情報や機密情報を扱うアプリケーションは、管理者権限を制限するなどより高いセキュリティレベルを要求される可能性がある。

- DevSecOps 環境を構築してもアプリケーションの脆弱性を完全に無くすことはできない。そのため、SIEM を利用した対策の実施や、軽微な脆弱性はトリアージを実施し対応の優先順位付けが必要になる。優先順位は各ツールによって決定され、順位が高いものから対処期限を設け修正作業を実施していく必要がある。

- 通常のウォーターフォール開発時のセキュリティチェックはOWASP ZAPのようなペネトレーションテストとセキュリティチェックシートのセルフチェックで実施されるが､DevSecOps環境ではソースコードの脆弱性やKubernetesクラスターの脆弱性、イメージ脆弱性など今までの診断では検出することのできなかった複数の脆弱性を可視化することができ、これはシステム全体の脆弱性を逓減することに繋がる。

以上が、DevSecOps環境を構築し、実際にサンプルアプリケーションデプロイした際の知見とそれに対する考察である。DevSecOpsはセキュリティを開発プロセスに組み込む有効な手法であるが、その導入や運用には課題やコストが伴うことが分かった。

7.2 セキュリティ対策への考察

全ての脆弱性を取りきれない状況でシステムをリリースし、セキュリティ対策を実施していく場合に考慮すべき点として、以下の考察を行った。

- DevSecOps 環境をイントラネット内に構築し、インターネットとの境界に L7 レベルでは WAF（Web Application Firewall）を、L4 レベルでは FW（Fire Wall）設置することで、外部からの攻撃を防ぐことが可能である。これにより、全ての脆弱性を取り除くことができない状況でも、ネットワークセキュリティを確保することができる。WAF は、HTTP トラフィックに対してアプリケーションレベルのセキュリティを提供するファイアウォールであり、FW は、外部と内部のネットワークトラフィックの間に障壁を提供するファイアウォールとなっており互いに補完的なセキュリティ機能を提供する。

- Hub＆Spoke のネットワークを構成することでネットワークレベルのセグメンテーションを実施する。具体的にはコンテナレジストリなど管理者が定期的にチェックする必要があるコンポーネントを Hub ネットワークに、開発者が利用する Kubernetes クラスターは各 Spoke ネットワークに配置することで万が一開発環境が攻撃により汚染された場合でも、他のネットワークに侵入できないようにする。

- SIEM の導入により、リアルタイムにセキュリティイベントを監視し、異常なパターンを検出することが可能となる。これにより、脆弱性が悪用される事態を早期に検出し、対応することができる。具体的な DevSecOps での SIEM の利用例としては、CI/CD パイプラインにおける自動化されたセキュリティが挙げられる。新たなコードがデプロイされる前の段階で適切なアプリケーションセキュリティテストを行い脆弱性になりうる異常を検知する。また、SIEM はログ情報を集約することで、セキュリティインシデントの原因分析や対策の検討に役立つ。

- 脅威モデリングの導入により､開発フェーズからセキュリティを考慮することが可能となる。これにより、開発初期段階でのセキュリティ対策の導入や、セキュリティ脆弱性の早期発見･修正が可能となる。これは､セキュリティ対策をリリース直前に行う場合に比べて､大幅なコスト削減に繋がる。例えばセキュリティチェックがリリースの最終段階にある場合、脆弱性の修正にはイメージやソースコードにまで遡る必要があり、また想定外のセキュリティホールが発見された場合はリリース時期の大幅な変更を余儀なくされる。このような将来の追加コストを未然に削減することができると考える。

以上のように、DevSecOps の導入により、開発プロセス全体を通じてセキュリティを確保することが可能となる。全ての脆弱性を取り除くことは難しいが WAF , FW やネットワーク分離、SIEM や脅威モデリングを応用することで対処可能であること考える。

第8章 おわりに

本調査研究において、OSS ベースの DevSecOps 環境に関する貢献が複数存在する。第一の貢献として、6.3 検証結果で示した OSS ベースの DevSecOps 環境の構築における結果から、システム脆弱性を逓減する可能性が高いことが確認された。具体的には、ソースコードスキャン、コンテナイメージのスキャン、アプリケーションのデプロイ後の脆弱性診断といった多角的なセキュリティ対策を通じて、脆弱性の早期発見と修正が可能であることが明示された。これらの措置は、特に既存のセキュリティ対策が不十分な場合や新たな脆弱性が発見された際に迅速な対応が求められるシナリオにおいて、有効であると考えられる。

第二の貢献として、7.1 構築結果への考察から OSS を活用することによる DevSecOps 環境のコスト及び工数に関する詳細な評価を提供した。OSS の採用は、導入初期コストを大幅に削減する可能性がある一方で、各コンポーネントに精通するためには継続的な学習と実践が不可欠であると考える。また、OSS コミュニティによる支援と迅速な修正が行われるため、セキュリティ維持においても高度な柔軟性と効率性が期待される。

第三の貢献としては、6.2 脆弱性診断の流れで示した開発から運用に至るまでのフェーズでのセキュリティ対策が体系的に構築された点である。Kubernetes の採用により、開発者はインフラストラクチャの煩雑な管理から解放され、より効率的にアプリケーションの開発に専念できる。また、開発段階で大きく 5 段階のセキュリティチェック項目を設け、運用段階では SIEM の運用を設けることで全体を俯瞰したセキュリティ対策が可能であると考える。これは DevSecOps の核心的理念である「開発と運用の統合」を具現化するものであり、その実現に向けての具体的なステップを提示した。

最後に、4.1 脅威分析手法の導入で示した STRIDE フレームワークによる脅威分析が実施された際、チーム内でのセキュリティスキル及び認識の向上が図られることを示した。この脅威モデリングにより、システムやアプリケーションに潜在するリスクが明確にされ、それに対する対策が計画的に行われる。特に、この手法を用いることで、セキュリティに対する一般的な理解や認識が低い開発者、運用者でも、具体的な脅威とリスクを明確に理解し、それに対する適切な対策を講じる能力が高まると考えられる。

本検証は OSS ベースの DevSecOps 環境における構築と運用について多角的な評価を行い、その成果として数々の有用な貢献を明らかにした。具体的には、環境の実用性、効果性、さらにはコストと工数に対する影響も詳細に調査・評価された。これらの成果は、DevSecOps の理論と実践における洞察を提供するものであり、特に実用性と効果性に焦点を当てた評価は、今後の DevSecOps 環境の設計と運用において極めて有用であると考えられる。さらに、本検証が提供するこれらの貢献は、DevSecOps が持つポテンシャルと、その実践における具体的な障壁を明確にした。その結果、開発、運用、そしてセキュリティの各分野での新たな取り組みに対して参考になる指針となると考えている。

近年の CNCF コミュニティやパブリッククラウド提供会社は複雑化したクラウド開発環境を鑑み、小規模なクラスター環境を用意し、開発ユーザーから Kubernetes などのインフラ機能を隠蔽することで開発の効率化を狙う製品も増えてきている。しかしエンタープライズ規模の開発環境をセキュアな状態で維持していくためにはアプリケーションやインフラストラクチャの知識や、セキュリティ、脆弱性についての知見、脅威分析手法の理解など幅広い知見が求められるのは今後も変わらないと考える。本研究が開発・運用・セキュリティそれぞれの取り組みを理解する手助けになれば幸いである。

参考文献

[1] DevOps
https://glossary.cncf.io/devops/

[2] アジャイル開発
https://grossary.cncf.io/agile-software-development

[3] DevSecOps
https://glossary.cncf.io/devsecops/

[4] 継続的インテグレーション（Continuous Integration）
https://glossary.cncf.io/continuous-integration/

[5] 継続的デプロイメント（Continuous Deployment）
https://glossary.cncf.io/continuous-deployment/

[6] セキュリティポリシー
https://learn.microsoft.com/en-us/azure/defender-for-cloud/security-policy-concept

[7] 脆弱性
https://www.soumu.go.jp/main_sosiki/joho_tsusin/security/basic/risk/11.html

[8] 脅威モデリング
https://owasp.org/www-community/Threat_Modeling

[9] 静的解析ツール
https://circleci.com/ja/blog/sast-vs-dast-when-to-use-them/

[10] 動的解析ツール
https://circleci.com/ja/blog/sast-vs-dast-when-to-use-them/

[11] SQL インジェクション
https://e-words.jp/w/ SQL インジェクション.html

[12] バッファオーバーフロー
https://e-words.jp/w/バッファオーバーフロー.html

[13] DREAD 分析
https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html#dread

[14] PASTA 分析
https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html#pasta

[15] STRIDE 分析
https://learn.microsoft.com/en-us/azure/security/develop/threat-modeling-tool-threats#stride-model

[16] Threat Modeling Tool feature overview
https://learn.microsoft.com/en-US/azure/security/develop/threat-modeling-tool-feature-overview#analysis-view
[17] SonarQube
https://docs.sonarqube.org/latest/
[18] Veracode
https://www.veracode.com/platform
[19] OWASP ZAP
https://www.zaproxy.org/
[20] Nmap
https://nmap.org/
[21] ソフトウェアコンポーネント分析（SCA）
https://owasp.org/www-community/Component_Analysis
[22] Black Duck
https://www.synopsys.com/software-integrity/security-testing/software-composition-analysis.html
[23] White Source
https://www.hitachi-solutions.co.jp/mend_sca/
[24] コンフィギュレーション管理データベース（CMDB）
https://www.atlassian.com/itsm/it-asset-management/cmdb
[25] Qualys
https://www.qualys.com/
[26] Rapid7
https://www.rapid7.com/
[27] Anchore
https://anchore.com/
[28] Trivy
https://github.com/aquasecurity/trivy
[29] ESLint
https://eslint.org/
[30] Chef InSpec
https://www.chef.io/products/chef-inspec
[31] Terraform
https://www.terraform.io/
[32] PCI DSS
https://www.jcdsc.org/pci_dss.php
[33] GDPR
https://www.ppc.go.jp/enforcement/infoprovision/EU/
[34] Linux
https://github.com/torvalds/linux
[35] UNIX OS
https://www.opengroup.org/membership/forums/platform/unix

[36] LXC
https://linuxcontainers.org/lxc/introduction/
[37] Docker
https://www.docker.com/
[38] Kubernetes
https://kubernetes.io/
[39] Borg
https://research.google/pubs/pub49065/
[40] CNCF
https://www.cncf.io/
[41] GitHub
https://github.com/
[42] Git
https://git-scm.com/
[43] Linus Torvalds
https://github.com/torvalds
[44] GitHub Actions
https://docs.github.com/en/actions
[45] GitHub Advanced Security
https://docs.github.com/en/get-started/learning-about-github/about-github-advanced-security
[46] Mac
https://www.apple.com/jp/mac/
[47] Windows
https://www.microsoft.com/ja-jp/software-download/windows11
[48] ARM
https://www.arm.com/
[49] Virtual Machine
https://www.vmware.com/topics/glossary/content/virtual-machine.html
[50] Node.js
https://nodejs.org/en
[51] Python
https://www.python.org/
[52] Java
https://www.java.com/en/
[53] Ruby
https://www.ruby-lang.org/ja/
[54] PHP
https://www.php.net/manual/ja/index.php
[55] Go
https://go.dev/
[56] Rust
https://www.rust-lang.org/

[57] .NET
https://dotnet.microsoft.com/en-us/download/dotnet-framework
[58] GitHub Enterprise Cloud
https://docs.github.com/en/get-started/onboarding/getting-started-with-github-enterprise-cloud
[59] GitHub Advanced Security Blog
https://github.blog/2021-03-30-github-advanced-security-security-overview-beta-secret-scanning-private-repos/
[60] Open Policy Agent (OPA)
https://www.openpolicyagent.org/
[61] Kyverno
https://kyverno.io/
[62] Rego
https://www.openpolicyagent.org/docs/latest/policy-language/
[63] Pod Security Policy (PSP)
https://kubernetes.io/docs/concepts/security/pod-security-policy/
[64] Pod Admission Policy
https://kubernetes.io/docs/concepts/security/pod-security-admission/
[65] Docker Registry
https://docs.docker.com/registry/
[66] Harbor
https://goharbor.io/
[67] Quay
https://www.projectquay.io/
[68] Docker Hub
https://hub.docker.com/
[69] Helm チャート
https://helm.sh/
[70] OCI アーティファクト
https://github.com/opencontainers/artifacts
[71] Azure Active Directory
https://azure.microsoft.com/en-ca/products/active-directory/
[72] Role-based Access Control
https://learn.microsoft.com/en-us/azure/role-based-access-control/overview
[73] Docker Content Trust
https://docs.docker.com/engine/security/trust/
[74] Notary
https://hub.docker.com/_/notary/
[75] OSS ベースの DevSecOps 環境構成図
[76] Pluggable Image Vulnerability Scanners for Harbor
https://blog.aquasec.com/container-image-vulnerability-scanner-harbor
[77] Severe Privilege Escalation Vulnerability in Kubernetes

https://blog.aquasec.com/kubernetes-security-cve-2018-1002105
[78] Kube-Bench: An Open-Source Tool for Running Kubernetes CIS Benchmark Tests
https://blog.aquasec.com/announcing-kube-bench-an-open-source-tool-for-running-kubernetes-cis-benchmark-tests
[79] ZAP - Webswing Usage
https://www.zaproxy.org/docs/docker/webswing/
[80] GRAFANA SUPPORT FOR PROMETHEUS
https://prometheus.io/docs/visualization/grafana/
[81] Visualizing security data with Canvas
https://www.elastic.co/blog/visualizing-security-data-canvas
[82] example-voting-app
https://github.com/dockersamples/example-voting-app
[83] DevSecOps lifecycle stages
https://learn.microsoft.com/en-us/azure/architecture/guide/devsecops/devsecops-on-aks#devsecops-lifecycle-stages
[84] What is CI/CD Pipeline?
https://medium.com/@nanduribalajee/what-is-ci-cd-pipeline-e2f25db99bbe
[85] DevSecOps on Azure Kubernetes Service (AKS)
https://learn.microsoft.com/enus/azure/architecture/guide/devsecops/devsecops-on-aks
[86] Cankar, M., Petrovic, N., Costa, J. P., Cernivec, A., Antic, J., Martincic, T., & Štepec, D. (2022). "Security in DevSecOps: Applying Tools and Machine Learning to Verification and Monitoring Steps". ACM Digital Library.
[87] Alonso, J., Piliszek, R., & Cankar, M. (2022). "Embracing IaC Through the DevSecOps Philosophy: Concepts, Challenges, and a Reference Framework". IEEE Xplore.
[88] Akbar, M. A., Smolander, K., Mahmood, S., & Alsand, A. (2022). "Toward successful DevSecOps in software development organizations: A decision-making framework". Information and Software Technology, 147.