セキュリティ製品の有効性検証の
検証結果について

令和２年４月

独立行政法人情報処理推進機構

目次

1. はじめに

経済産業省は 2017 年 12 月に「産業サイバーセキュリティ研究会」を設置し、ワーキンググループ 3（サイバーセキュリティビジネス化）において、有効性検証を通じ日本発のサイバーセキュリティ製品・サービスのマーケット・イン促進に資するサイバーセキュリティ検証基盤（以下、検証基盤）の構築を目指すとしている。

経済産業省の委託を請け、独立行政法人情報処理推進機構(以下、IPA)では、検証基盤の在り方を検討する「サイバーセキュリティ検証基盤構築に向けた有識者会議[1]（以下、有識者会議）」を設置した。有識者会議の検討において、検証基盤の対象製品を日本発のスタートアップ製品とし、その優れた特長について検証する仕組みとして具体化すること、また今年度は、脅威・脆弱性の可視化及び IT 資産管理に係る製品分野を検証対象とすること、を方針とした。

具体的な検証方式として経済産業省の計画では、(1)専門家による客観的な「セキュリティ製品の有効性検証」と(2)利用者の「実環境における試行検証」の 2 種類を実施することとしている。そこで有識者会議は、この計画に基づいた検証体制や検証方法等の実施案を検討し、さらにその効果や課題を明らかにするため試行検証を行うこととした。まず、対象となるセキュリティ製品を公募し、上記二つの検証方法それぞれ1製品、計 2 製品を選定して、試行検証の題材とし、IPA が事務局となって実際に試行検証を実施した。

以下、ビジョナル・インキュベーション株式会社「yamory」を対象に実施した「セキュリティ製品の有効性検証」の検証結果について示す。

2. 対象製品「yamory」について

2.1. 概要

yamory は、利用している OSS を自動抽出し、脆弱性を検出、管理するサービスである。

yamory は NVD 等の公開されている脆弱性データベースから収集した OSS に関する脆弱性情報を用いて、yamory 独自の脆弱性情報（脆弱性データベース）を構築している。

yamory の脆弱性データベース内の情報と、ビルドシステム[2]/パッケージ管理システム[3]で管理している OSS のバージョン情報を照合することで、OSS に含まれる脆弱性を検出する。

---

[1] IPA サイバーセキュリティ検証基盤構築に向けた有識者会議
https://www.ipa.go.jp/security/economics/kensyokiban2019.html

[2] ビルドシステム：プログラムのソースコードから、実際に動くプログラムを作成するツールやシステム。

[3] パッケージ管理システム：プログラムの実行ファイルや設定ファイル、ライブラリ等を管理するツールやシステム。

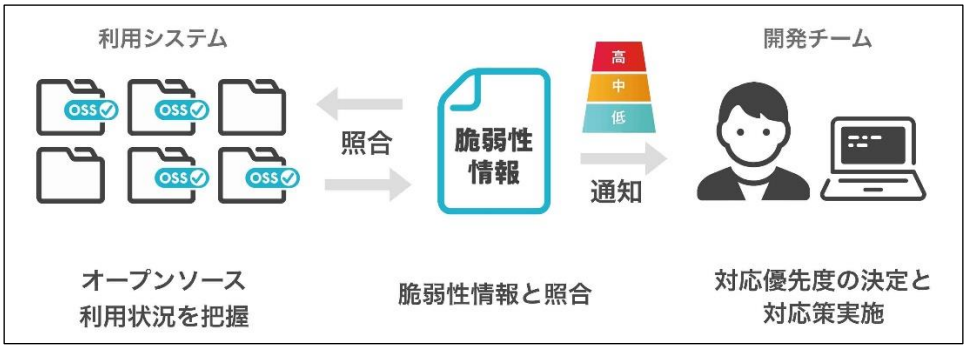

図 1 yamory の利用イメージ

https://www.bizreach.co.jp/pressroom/pressrelease/2019/0827.html より転載

2.2. OSS について

2.2.1. OSS とは

OSS とは Open Source Software の略称であり、人間が読み取れるソースコードを使用、研究、再利用、修正、拡張、そのソフトウェアのユーザーによる再配布可能なソフトウェアを指す。

（https://dodcio.defense.gov/Open-Source-Software-FAQ/より引用）

2.2.2. OSS 利用の背景と問題

昨今の IT/Web システムの開発では、OSS を有効活用することが主流となっている。その背景には、ゼロからシステム開発する場合に比べ、既存の OSS を組み合わせた方が、効率良く開発できることが挙げられる。

一方で、OSS 利用時の問題点として、利用している OSS を開発者自身が把握できていないことや、商業製品のようなサポートを受けることができないため、OSS の脆弱性情報、最新バージョン等の情報は、開発者自身が積極的に収集する必要がある。

システムで利用している個々の OSS を把握し、日々脆弱性が存在しないかをチェックして対応することは困難であり、OSS の管理、脆弱性への対応が課題と考えられている。

2.3. 製品特徴

利用している OSS の自動抽出機能や、検出した脆弱性の優先度を自動的に分類する機能が特徴として挙げられる。

また、脆弱性対応の役割ごとに、「開発チーム」、「セキュリティチーム」があり、開発チームは個々の開発システムを管理する役割であるのに対し、セキュリティチームは全システムを横断的に管理する役割となっている点も特徴である。

3. 検証項目

開発者および CSIRT 等のセキュリティを管理するチームの観点で OSS の脆弱性検査・管理をする上で有効と思われる下記の項目について検証した。

3.1. 検査品質

脆弱性を検査するツールは、脆弱性をもれなく検出できることで品質が高いと言える。

OSS は、内部で他の OSS を利用していることが多く、依存関係が存在する。そのため、依存関係のある OSS に関しても脆弱性が検出可能なのか、また最新の OSS だけでなく、ひと昔前の OSS でも検出可能なのか、新しい脆弱性が公表された際に検出可能なのか、といった観点から以下を検証項目とした。

検証項目 G1-1：依存関係のある OSS の脆弱性が検出可能か
検証項目 G1-2：古い OSS の脆弱性が検出可能か
検証項目 G1-3：新規脆弱性が公表された場合の検出までのリードタイムはどの程度か

3.2. 脆弱性付加情報

脆弱性が検出された場合、対応可否の判断および対応方法を把握するために、脆弱性の詳細な内容について確認する必要がある。

多数の脆弱性が検出された場合、どこから対策すべきか見えず、対策が後手になる、または対応が進まないという状況に陥りがちである。脆弱性の詳細情報や、脆弱性のリスクにより、優先して対応すべきものが判断できれば、効果的に対策を進めることができる。

脆弱性を検出した後、スムーズに脆弱性対応に着手するための情報が充実しているか、という観点から、以下を検証項目とした。

検証項目 G2-1：検出した脆弱性に関する情報が充実しているか
検証項目 G2-2：検出した脆弱性の対応優先度を容易に判断可能か

3.3. 脆弱性対応管理

脆弱性が検出されたとしても、修正対応を行わなければ意味がなく、検出から脆弱性対応の完了までを管理することが大切である。

ツールによってはスキャン結果が表示されるのみで管理機能が無いものもあり、スキャン結果を出力し Excel で管理、または別のツールに取り込む等の作業が必要な場合もある。

同一ツール上で脆弱性検出から対応管理までできることで、より効率的な管理ができるものと考え、以下を検証項目とした。

検証項目 G3-1:脆弱性の対応管理機能を備えているか

3.4. 複数システムの品質管理

OSS の脆弱性を管理するためには、利用している OSS を把握することが必要であるが、組織では複数のシステムを運用していることが多い上、システムの開発構成を一元管理できていない等、個々のシステムで利用している OSS を把握することは困難な場合がある。

また、脆弱性の管理においても、個々のシステムを別々に管理した場合、管理タスクの増加による管理漏れ等が原因で、特定システムの脆弱性対応ができていないことに気が付かず、そこからインシデントに発展する恐れもある。

そのため、CSIRT 等のセキュリティを管理するチームが、複数のシステムの脆弱性を横断的に管理し、開発チームの脆弱性対応を推進できることが望ましいことから、以下を検証項目とした。

検証項目 G4-1:組織内で使用されている OSS の一覧表示が可能か
検証項目 G4-2:組織内で検出されている脆弱性の一覧表示が可能か
検証項目 G4-3:セキュリティ管理チームの脆弱性分析を基に開発チームへの対応指示が可能か

3.5. 運用容易性

ツールを導入しても、使い勝手の面で難があれば敬遠されることも考えられる。

限られたリソースの中で、なるべく手間を掛けずに検査を実施可能か、UI[4]が明瞭であり使いやすいか、脆弱性が検出された際に通知があるか等、実際の運用時を想定した観点から、以下を検証項目とした。

検証項目 G5-1:検査を自動的に、繰り返し実施可能か
検証項目 G5-2:ダッシュボードにおいて検出状況が容易に確認可能か
検証項目 G5-3:脆弱性検出時や対応指示入力時にメール等での通知機能があるか

3.6. 導入容易性

ツールを導入するにあたり、導入前に多くの手順が必要、現在の環境に影響を与える可能性がある、導入のために環境自体を変える必要がある等の場合には、検討時に選定の対象とならないこともある。

導入するにあたって障壁となる部分がいかに少ないか、という観点から以下を検証項目とした。

[4] UI:ユーザーインターフェース。ユーザーとコンピュータ、ソフトウェアで情報をやり取りする仕組み。

検証項目 G6-1:利用している OSS 情報を新たに作成することなく利用可能か
検証項目 G6-2:リモートリポジトリ・ローカルリポジトリのスキャンに対応可能か
検証項目 G6-3:エージェントをインストールすることなく利用可能か

4. 検証環境

今回の検証のため構築した環境は以下の通り。
実際の運用環境よりもシンプルな構成とした。

表 1 使用した OS、ソフトウェアとバージョン情報

	名称	バージョン
OS	Ubuntu[5]	19.10
ビルドシステム/パッケージ管理システム	Gradle[6]	6.1.1
アプリケーションフレームワーク	Struts2[7]	2.3.14.1
CI ツール	Jenkins[8]	2.204.1

5. 検証結果

5.1. 検査品質

5.1.1. 検証項目 G1-1:

依存関係のある OSS の脆弱性が検出可能か

検証結果:

Struts2(ver2.3.14.1)に対して脆弱性の検出精度を検証した結果、依存関係のある OSS の脆弱性を含め、網羅的に検出できた。

検証内容詳細:

検証で利用する OSS の Struts2(ver2.3.14.1)が保有している脆弱性に対して、検出精度を確認する。コンパイル系言語については、リモートリポジトリ(GitHub)でのスキャンと、ローカルリ

[5] Ubuntu:Debian GNU/Linux をベースとしたオペレーティングシステム。
[6] Gradle:オープンソースのビルド自動化システム。https://gradle.org/
[7] Struts2:オープンソースの Java Web アプリケーションフレームワーク。https://struts.apache.org/
[8] Jenkins:オープンソースの継続的インテグレーションツール。https://jenkins.io

ポジトリでコマンド実行してスキャンした場合の検出精度に違いがあり、検出精度の高いローカルリポジトリでコマンドを実行した結果を検証した。

NVD に掲載されている脆弱性と yamory で検出した脆弱性結果から調査した結果を照合する。

Struts2（ver2.3.14.1）の OSS 依存関係は以下の図 2 の通りである。

```
¥--- org.apache.struts:struts2-core:2.3.14.1
     +--- org.apache.struts.xwork:xwork-core:2.3.14.1
     |    +--- org.apache.commons:commons-lang3:3.1
     |    +--- ognl:ognl:3.0.6
     |    |    ¥--- javassist:javassist:3.11.0.GA
     |    +--- asm:asm:3.3
     |    ¥--- asm:asm-commons:3.3
     |         ¥--- asm:asm-tree:3.3
     |              ¥--- asm:asm:3.3
     +--- org.freemarker:freemarker:2.3.19
     +--- ognl:ognl:3.0.6 (*)
     +--- commons-fileupload:commons-fileupload:1.2.2
     ¥--- commons-io:commons-io:2.0.1
（コマンド「Gradle Dependencies」より取得）
```

図 1 Struts2（ver2.3.14.1）の OSS 依存関係

表 2 は、NVD に掲載されている脆弱性情報と yamory で検出した結果を照合した結果である。

NVD に掲載されている Struts2（ver2.3.14.1）の脆弱性（ソフトウェア列が org.apache.struts:struts2-core:2.3.14.1 の箇所）について網羅的に検出できていることを確認した。また、依存関係のあるソフトウェア「commons-fileupload:commons-fileupload:1.2.2」の脆弱性についても検出できていることを確認した。

表 1 NVD 掲載の脆弱性情報と yamory 検出結果の照合

NVD に掲載されている脆弱性情報		yamory 検出結果	
No.	CVE-ID	検出有無	ソフトウェア
1	CVE-2013-0248	○	commons-fileupload:commons-fileupload:1.2.2

NVD に掲載されている脆弱性情報		yamory 検出結果	
No.	CVE-ID	検出有無	ソフトウェア
2	CVE-2014-0050	○	commons-fileupload:commons-fileupload:1.2.2
3	CVE-2016-1000031	○	commons-fileupload:commons-fileupload:1.2.2
4	CVE-2016-3092	○	commons-fileupload:commons-fileupload:1.2.2
5	CVE-2013-2115	○	org.apache.struts:struts2-core:2.3.14.1
6	CVE-2013-2134	○	org.apache.struts:struts2-core:2.3.14.1
7	CVE-2013-2135	○	org.apache.struts:struts2-core:2.3.14.1
8	CVE-2013-2248	○	org.apache.struts:struts2-core:2.3.14.1
9	CVE-2013-2251	○	org.apache.struts:struts2-core:2.3.14.1
10	CVE-2013-4310	○	org.apache.struts:struts2-core:2.3.14.1
11	CVE-2013-4316	○	org.apache.struts:struts2-core:2.3.14.1
12	CVE-2014-0094	○	org.apache.struts:struts2-core:2.3.14.1
13	CVE-2014-0112	○	org.apache.struts:struts2-core:2.3.14.1
14	CVE-2014-0113	○	org.apache.struts:struts2-core:2.3.14.1
15	CVE-2014-0116	○	org.apache.struts:struts2-core:2.3.14.1
16	CVE-2014-7809	○	org.apache.struts:struts2-core:2.3.14.1
17	CVE-2015-5169	○	org.apache.struts:struts2-core:2.3.14.1
18	CVE-2015-5209	○	org.apache.struts:struts2-core:2.3.14.1
19	CVE-2016-0785	○	org.apache.struts:struts2-core:2.3.14.1
20	CVE-2016-2162	○	org.apache.struts:struts2-core:2.3.14.1
21	CVE-2016-3081	○	org.apache.struts:struts2-core:2.3.14.1
22	CVE-2016-3082	○	org.apache.struts:struts2-core:2.3.14.1
23	CVE-2016-3090	○	org.apache.struts:struts2-core:2.3.14.1
24	CVE-2016-3093	○	org.apache.struts:struts2-core:2.3.14.1
25	CVE-2016-4003	○	org.apache.struts:struts2-core:2.3.14.1
26	CVE-2016-4436	○	org.apache.struts:struts2-core:2.3.14.1
27	CVE-2016-4461	○	org.apache.struts:struts2-core:2.3.14.1
28	CVE-2017-12611	○	org.apache.struts:struts2-core:2.3.14.1
29	CVE-2017-5638	○	org.apache.struts:struts2-core:2.3.14.1
30	CVE-2017-9787	○	org.apache.struts:struts2-core:2.3.14.1
31	CVE-2017-9791	○	org.apache.struts:struts2-core:2.3.14.1
32	CVE-2017-9793	○	org.apache.struts:struts2-core:2.3.14.1
33	CVE-2017-9804	○	org.apache.struts:struts2-core:2.3.14.1

NVD に掲載されている脆弱性情報		yamory 検出結果	
No.	CVE-ID	検出有無	ソフトウェア
34	CVE-2017-9805	○	org.apache.struts:struts2-core:2.3.14.1
35	CVE-2018-11776	○	org.apache.struts:struts2-core:2.3.14.1
36	CVE-2018-1327	○	org.apache.struts:struts2-core:2.3.14.1

5.1.2. 検証項目 G1-2:

古い OSS の脆弱性の検出が可能か

検証結果:

NVD 等の脆弱性データベースに掲載されている OSS の脆弱性については検出可能である

検証内容詳細:

本件は製品ベンダへのヒアリングによる評価とした。

NVD を主とした脆弱性データベースの中から、アプリケーションレイヤ[9]、且つパッケージ管理システムの公式リポジトリにホストされている OSS パッケージの脆弱性情報を収集し、yamory のデータベース上に反映しており、NVD 等の脆弱性データベースに掲載されている OSS の脆弱性であれば、検出可能である。

5.1.3. 検証項目 G1-3:

新規脆弱性が公表された場合の検出までのリードタイムはどの程度か

検証結果:

脆弱性が公表されてから yamory の脆弱性データベースに登録されるまでのリードタイムは、緊急性に応じて、即時から 3 日程度である。

また、過去にスキャンした OSS の新規脆弱性が yamory の脆弱性データベースに登録された場合、再スキャンを実施しなくても新規脆弱性の検出が可能である。

なお、脆弱性が NVD 等の脆弱性データベースに掲載される前の段階、または掲載されてから検出されるまでの間に、OSS のコミュニティ等で脆弱性の疑いのある情報を得ていた場合、組織で使用している OSS を検索することで、脆弱性のある OSS を使用しているかを即座に把握できるため、yamory で検出できない状態でも対応を検討できる。

[9] アプリケーションレイヤ:システムを構成する階層のひとつ。システムは大まかにハードウェア、OS、ミドルウェア、アプリケーションの 4 つの階層に分けられる。

※OSS の検索機能については「5-4.複数システムの品質管理」の検証項目 G4-1 を参照のこと。

検証内容詳細:

本件は製品ベンダへのヒアリングによる評価とした。

新規脆弱性が NVD 等の脆弱性データベースに公表された後、緊急性の高い脆弱性については、yamory の脆弱性データベースに通常よりも短期間で登録している。一般的な脆弱性については実績値として 1 日から最長 3 日程度で脆弱性データベースに登録するとしている。

5.2. 脆弱性付加情報

5.2.1. 検証項目 G2-1:

検出した脆弱性に関する情報が充実しているか

検証結果:

検出された脆弱性のリスク、概要や対応方法、CVSS[10]、PoC[11]の有無、関連 URL、CPE 等の情報が閲覧可能であり、脆弱性の関連情報を把握可能なものとなっている。

PoC が流通している場合、脆弱性を突いた攻撃を受ける可能性が高くなるため、PoC の有無まで確認できるということは非常に有益であると言える。

検証内容詳細:

図 2 は検出した脆弱性の情報を確認できる画面である。

[10] CVSS:共通脆弱性評価システム(Common Vulnerability Scoring System)。 情報システムの脆弱性に対するオープンで汎用的な評価手法のひとつで、基本評価基準、現状評価基準、環境評価基準の 3 つ指標を用いて評価する。https://www.ipa.go.jp/security/vuln/CVSS.html

[11] PoC:Proof of Concept(概念実証)。セキュリティの分野では概念実証コード等とも呼ばれ、脆弱性を悪用した攻撃が実際に有効であることを示すプログラムを指す。

図 2 脆弱性情報

セキュリティチームからの対応方針、脆弱性によるリスク、対応方法が確認できる。

対応方法については、アップデートを促すものが基本であるが、図 2 の脆弱性のように発生条件がある場合には、その詳細が確認できるものもある。[12]

図 3 のように詳細情報として、脆弱性の種類、公表元や関連 ID、概要、更に下部のタブを切り替えることで、CVSS、PoC の URL、参考・関連情報 URL、CPE を確認できる。

[12] 掲載の脆弱性は今回の検証環境外で検出した例である。

図 3 脆弱性詳細画面

また、関連情報 URL には様々なサイトへのリンクが掲載されており、NVD や JVN 等の関連情報を参照できる。

5.2.2. 検証項目 G2-2:

検出された脆弱性の対応優先度を容易に判断可能か

検証結果:

yamory のオートトリアージ機能により、自動的に対応優先度が分類されるため、優先的に対応すべき脆弱性を容易に判断できる。多くの脆弱性を検出した場合でも、脆弱性の分析に時間を掛けることなく、最優先で解決すべき脆弱性を把握できるため、効率的に対応を進めることができる。

検証内容詳細:

yamory には、オートトリアージ機能と呼ばれる各脆弱性の対応優先度を自動的に分類する仕組みがあり、スキャンを行うだけで対応優先度が設定される。

図 4 はスキャン後のダッシュボード上の表示である。

図 4 ダッシュボードの表示

優先度が高い順に、「Immediate」、「Delayed」、「Minor」、「None」として分類されており、それぞれの判断基準は図 5 の通りである。

分類名	想定されるリスク	対応目安	判断基準
Immediate	即時に影響を受ける可能性がある	検出された当日中に、影響有無の調査、対処を推奨します	危険な脆弱性 公開サービス 攻撃コード（PoC）あり にあてはまる脆弱性
Delayed	攻撃コードの流通次第で、即時に影響を受ける可能性がある	2週間以内に影響有無の調査、対処を推奨します	危険な脆弱性 公開サービス にあてはまる脆弱性
Minor	外部から直接攻撃が可能な状態ではないが、二次被害として影響を受ける可能性がある	月に1回程度の頻度で、影響有無の調査、対処を推奨します	**Immediate** または **Delayed** にあてはまらない脆弱性であり、かつ CVSS が Critical または High と判断された脆弱性
None	被害につながる可能性が低い	定期的なメンテナンス等で影響有無の調査、対処を推奨します	**Immediate** または **Delayed** にあてはまらない脆弱性であり、かつ CVSS が Medium 以下と判断された脆弱性

図 5 オートトリアージの判断基準

https://yamory.io/docs/auto-triage より転載

5.3. 脆弱性対応管理

5.3.1. 検証項目 G3-1:

脆弱性の対応管理機能を備えているか

検証結果:

脆弱性を検出してから、対応完了までのステータス管理、および対応に関するタイムラインを管理可能な機能を備えていることで、個々の脆弱性の対応状況を把握できる。

検証内容詳細:

検出した脆弱性の対応を行う際には、脆弱性の情報画面から、対応ステータスを変更できる。ステータスは、「対応を開始する」、「修正しないで完了(脆弱性あり/不明)」、「修正しないで完了(脆弱性なし)」の中から選択する。

図 6 対応ステータスの変更

今回は「対応を開始する」を選択し、対応コメントを入力する。

図 7 コメント入力欄

入力したコメントは、脆弱性詳細画面のタイムラインに表示されており、脆弱性に対するやり取りを確認できる。

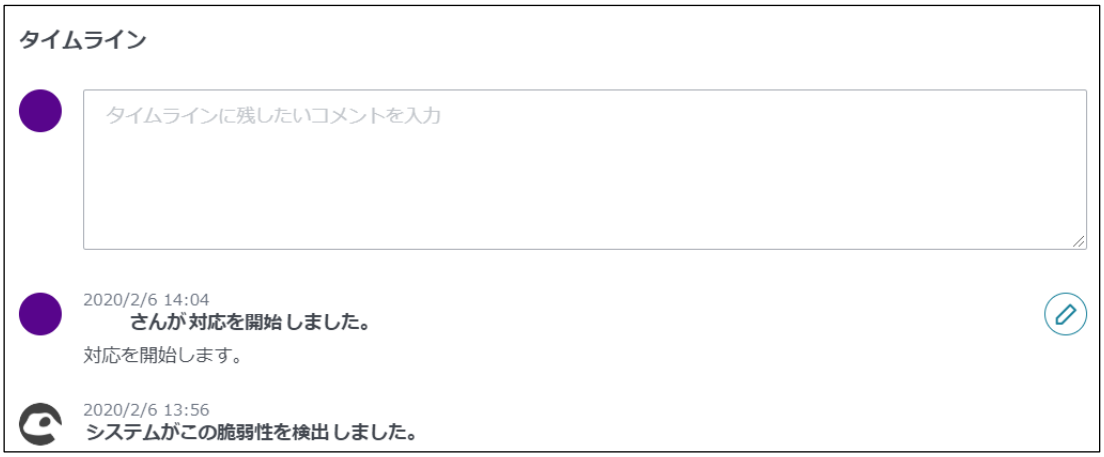

図 8 タイムラインへのコメント表示

対応を開始すると、脆弱性の対応ステータスが「対応中」に変化しており、対応状況を確認できる。

図 9 「対応中」のステータス

対応完了後は、手動でステータスを「完了」にする必要はなく、再スキャンを実行した結果、脆弱性が検出されない場合に自動的に「完了」となる。

図 10 「完了」のステータス

上記により、脆弱性を検出してから、対応完了までのステータス管理、および対応に関するタイムラインを管理可能な機能を備えていることで、個々の脆弱性の対応状況を把握できる。

5.4. 複数システムの品質管理

5.4.1. 検証項目 G4-1:

組織内で使用されている OSS の一覧表示が可能か

検証結果:

セキュリティチームのメンバーは、組織内で使用している OSS の一覧表示が可能である。また、キーワードで検索でき、OSS のコミュニティ等で話題になった脆弱性のある OSS を、保有システムで利用しているかを容易に確認できる。

検証内容詳細:

セキュリティチームは組織の全システムで使用している OSS を確認できる。

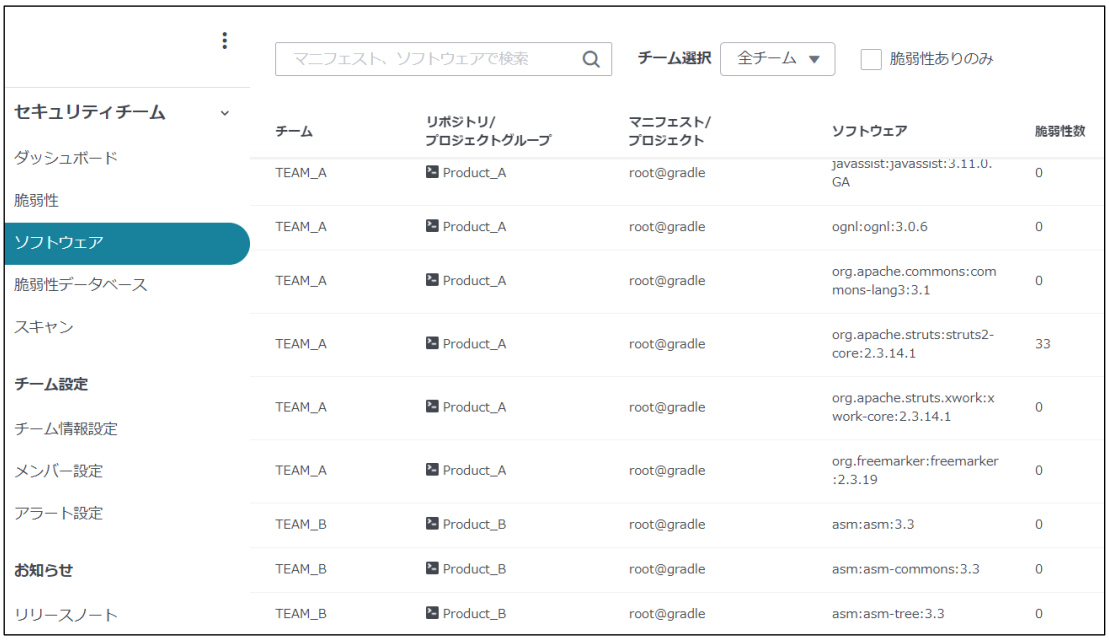

図 11 ソフトウェア一覧表示

検索フィールドにキーワードを入力し、組織が使用している OSS を検索できる。

検索機能を利用することで、OSS のコミュニティ等で話題になった脆弱性のある OSS を、保有システムで利用しているかを容易に確認できる。

開発チーム単位への切り替えや、脆弱性が存在する OSS のみ確認することもできる。

5.4.2. 検証項目 G4-2:

組織内で検出されている脆弱性の一覧が表示可能か

検証結果:

セキュリティチームのメンバーは、組織内の複数のシステムで検出されている脆弱性一覧を確認できる。

CVE-ID 等で脆弱性を指定して検索することで、該当の脆弱性が検出されているシステムを把握できる。

検証内容詳細:

セキュリティチームのメンバーは、組織の全システムで検出されている脆弱性を確認できる。

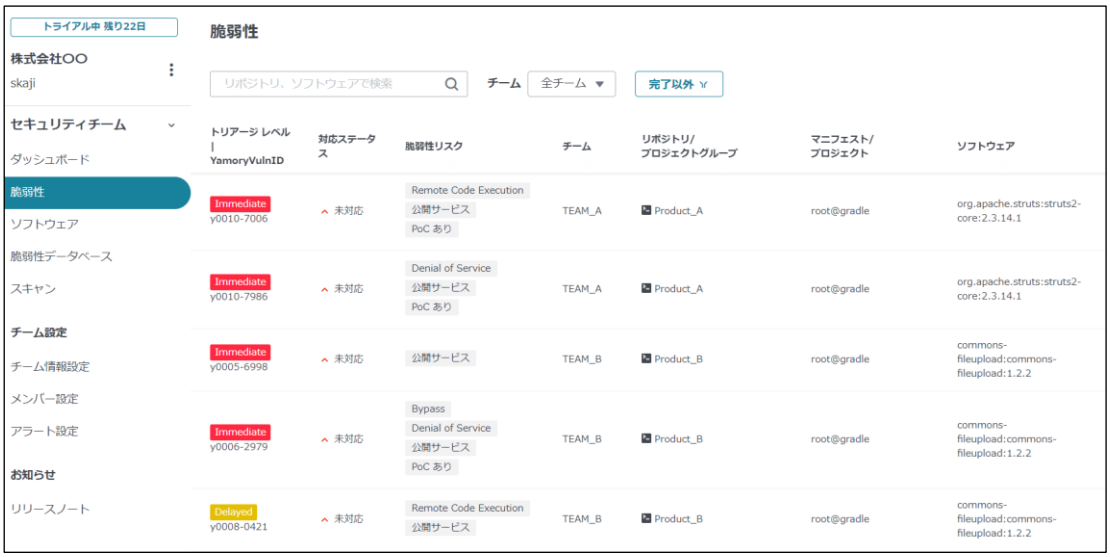

図 12 脆弱性一覧表示

画面上部の検索フィールドから、リスクの高い脆弱性の CVE-ID 等で検索することで、該当の脆弱性を保有するシステムを確認することもできる。

また、セキュリティチームが使用できる「脆弱性データベース」も存在する。

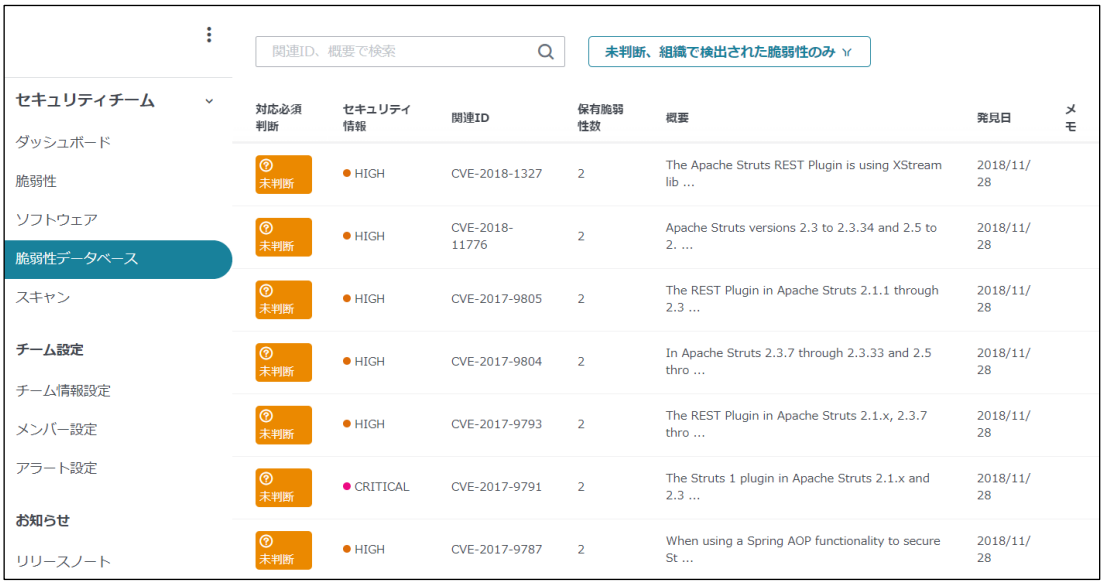

図 13 脆弱性データベース表示

yamory のデータベースに登録されている、組織が保有していない脆弱性も全て確認可能となっており、2020 年 2 月現在、約 136,000 件の脆弱性が登録されている。

各脆弱性を選択することで、脆弱性の詳細画面を確認できる。

5.4.3. 検証項目 G4-3:

セキュリティ管理チームの脆弱性分析を基に開発チームへの対応指示が可能か

検証結果:

セキュリティチームは、組織独自の基準に基づき、オートトリアージ機能で決定されたトリアージレベルの変更や、脆弱性の対応方針を開発チームへ周知できる。

検証内容詳細:

セキュリティチームは、各脆弱性のトリアージレベルの変更と、対応方針を入力できる。

図 14 トリアージレベルの選択と対応方針の入力

開発チームは、対応方針のコメントを脆弱性詳細の「セキュリティチームからの対応方針」で確認できる他、「危険な脆弱性」として判断され、「Immediate」となった脆弱性は、開発チームにメール等で通知が行われる。

図 15 セキュリティチームからの対応方針の表示

オートトリアージ機能により、優先度が自動設定されるため、何を優先して対応すべきか、開発チーム側でもある程度把握できることが yamory の特徴と言えるが、組織独自の基準により、優先度は異なるものと考えられる。

その場合でも、セキュリティチームが優先度を再設定し、対応指示ができる。

5.5. 運用容易性

5.5.1. 検証項目 G5-1:

検査を自動的に、繰り返し実施可能か

検証結果:

GitHub リポジトリに対するスキャンは日次で自動実行される。ローカルリポジトリ上でのコマンド実行によるスキャンは CI ツールに組み込むことで定期的に実行できる。yamory のコマンド実行によるスキャンは高速で実行される為、CI ツールで繰り返しビルドを行う場合でも組み込み易くなっている。

検証内容詳細:

GitHub リポジトリに対するスキャンは日次で自動実行されることを確認した。

ローカルリポジトリ上でのコマンド実行によるスキャンについては、CI ツールの「Jenkins」(2.204.1)を使用することで確認した。

図 16 Jenkins を用いたスキャン用ジョブの作成

yamory から発行されるスキャンコマンドをローカルリポジトリで実行するよう、シンプルなジョブ[13]を作成し、本ジョブが定期的に実行されるよう設定しておくだけで、定期的にスキャンを実行できる。

yamory はソースコードをスキャンすることなく、パッケージ管理システムで管理された OSS のバージョン情報を使用してスキャンを行うため、スキャンが高速で完了する。

スキャンの処理時間が短いため、CI ツールで繰り返しビルドを行う場合でも組み込み易いということが利点と言える。

5.5.2. 検証項目 G5-2:

ダッシュボードにおいて検出状況が容易に確認可能か

検証結果:

ダッシュボード上で優先度別の脆弱性件数や、脆弱性件数の推移グラフが確認できる。

セキュリティチームは各チームの対応状況推移が表示されるため、脆弱性を多く抱えるシステムを容易に把握できる。

検証内容詳細:

開発チームの「ダッシュボード」では、チーム内で検出されている脆弱性の件数が優先度別に表示される。

また、危険度の高い脆弱性である「Immediate」、「Delayed」の件数の推移をグラフで確認でき

[13] ジョブ:コンピュータが処理する仕事の単位。

る。

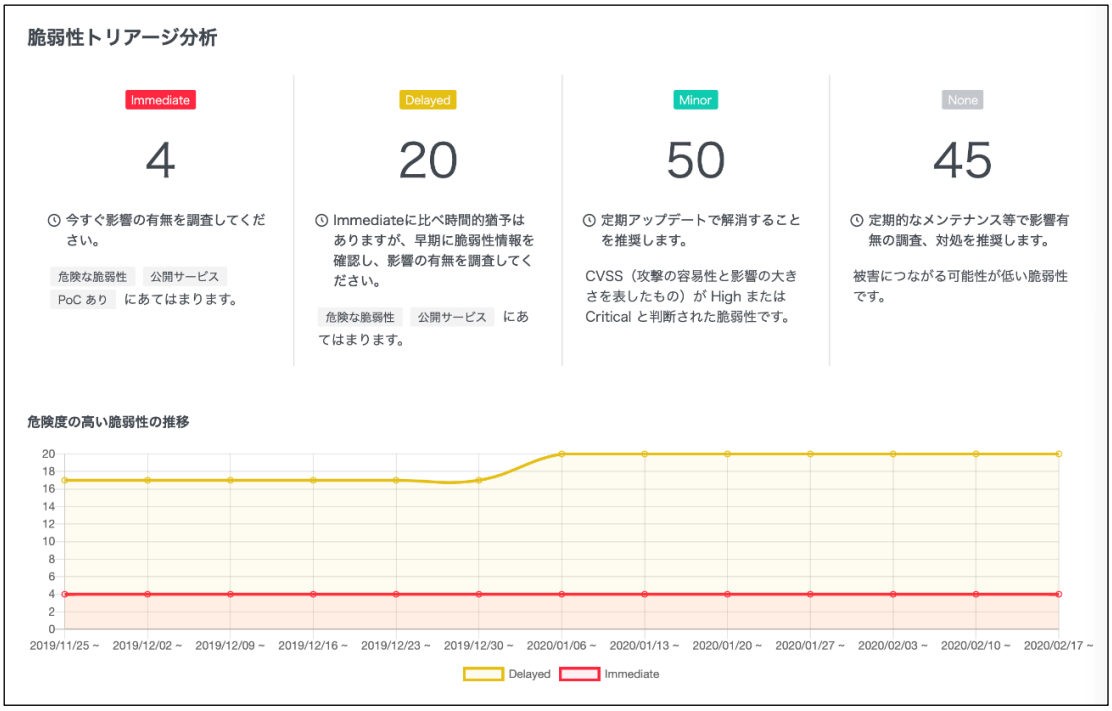

図 17 ダッシュボードに表示される脆弱性件数と推移グラフ

（ビジョナル・インキュベーションより提供）

セキュリティチームでは、優先度別の脆弱性件数が全システムの合算値で表示される。

グラフについては「Immediate」、「Delayed」それぞれの対応状況の推移をチーム別に確認できる。

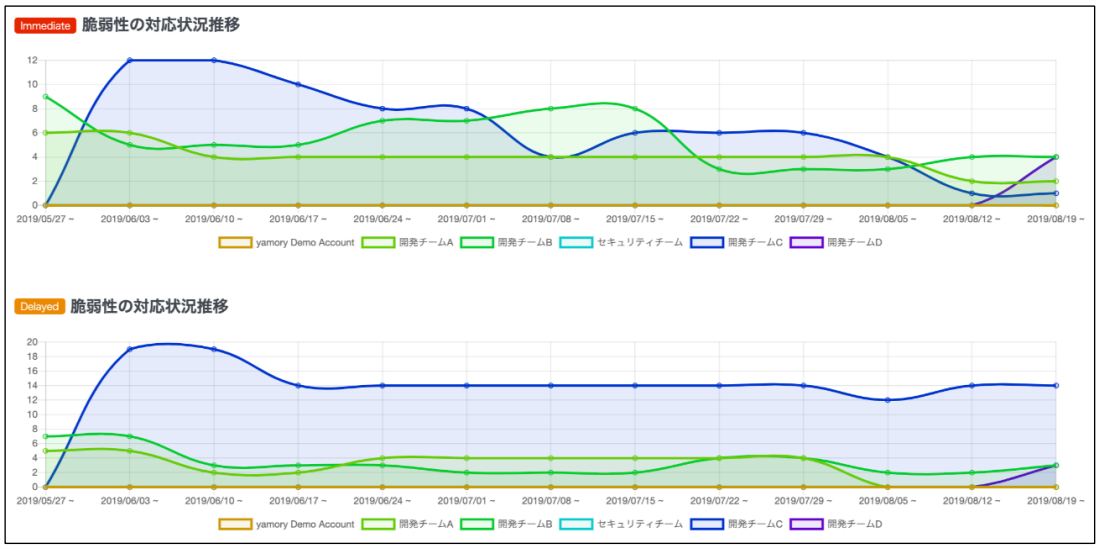

図 18 セキュリティチームで表示される対応状況推移グラフ

（ビジョナル・インキュベーションより提供）

5.5.3. 検証項目 G5-3:

脆弱性検出時や対応指示入力時にメール等での通知機能があるか

検証結果:

「Immediate」、「Delayed」の脆弱性が検出された場合、メールでの通知機能や、Slack との連携機能による通知が可能で、yamory 上で確認せずとも脆弱性が検出されたことを把握できる。

またヒアリングにより、Slack 以外のコミニューケーションツールとの連携は、市場動向を踏まえながら拡張していくことを伺った。

検証内容詳細:

セキュリティチームや開発チームのメンバーは、新規脆弱性が検出された場合、登録しているメールアドレス宛に通知が行われる。

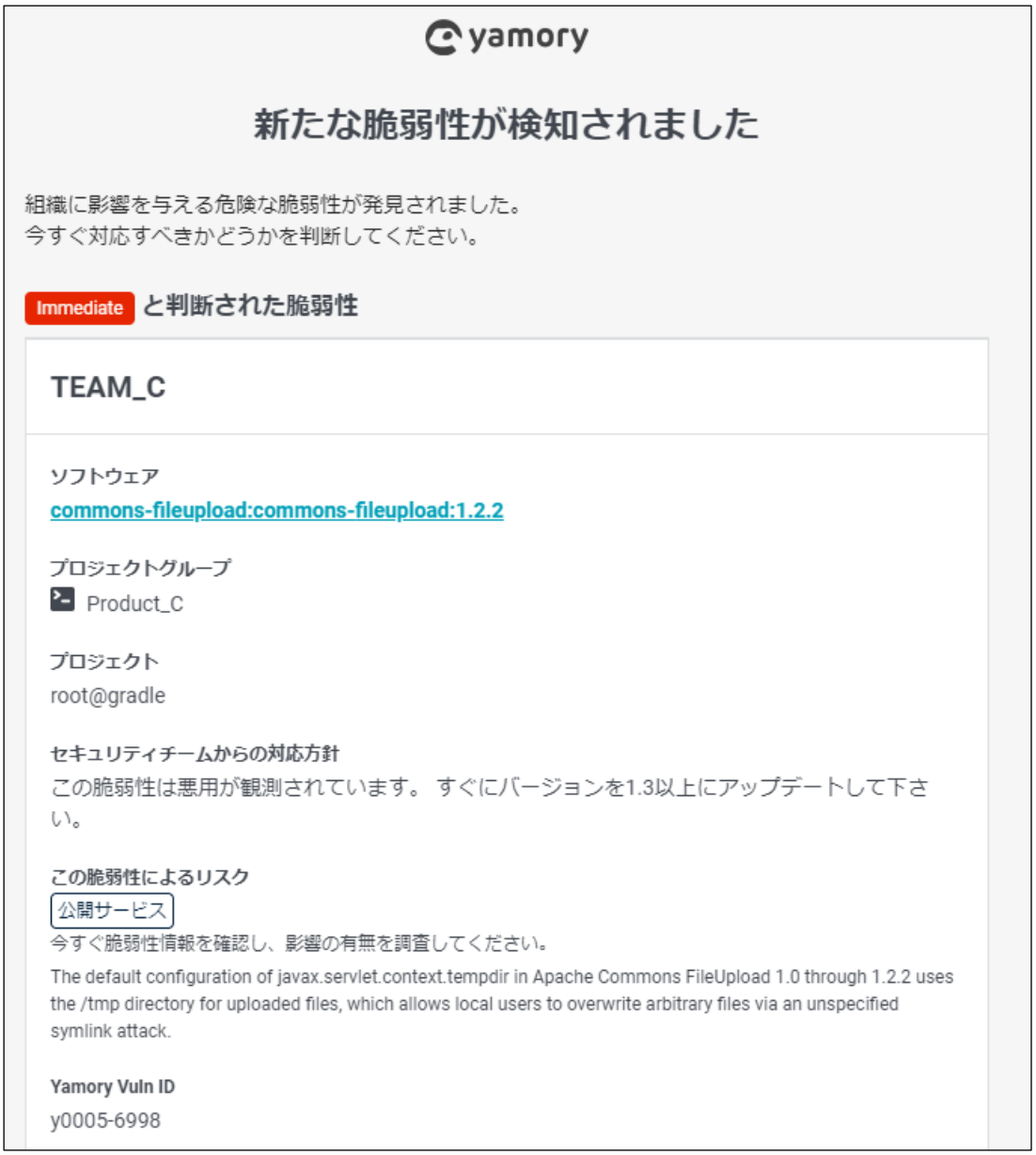

図 19 メールでの脆弱性通知

また、組織のコミュニケーションツールとして利用している場合には、図 20 のように Slack で通知を受けることができる。

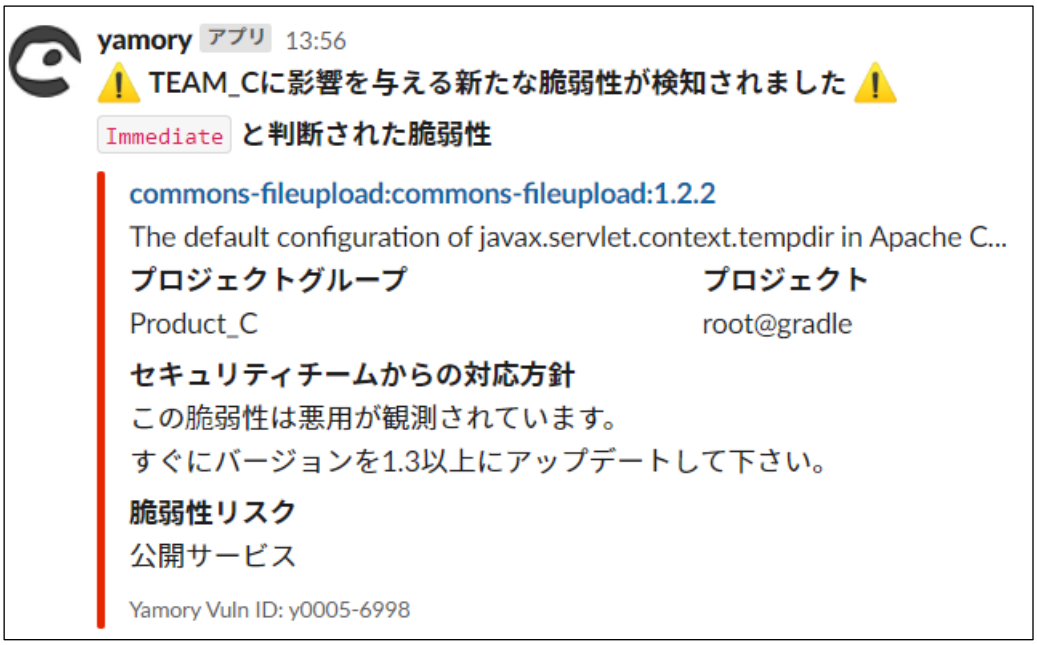

図 20 Slack での脆弱性通知

新規脆弱性検出時だけではなく、セキュリティチームがトリアージレベルを「危険な脆弱性」と判断したことで「Immediate」となった場合、該当脆弱性があるシステムの開発チームには、通知が届く仕組みとなっている。

また、メールでは週に 1 回、週間報告として脆弱性の前週比、および今週検出された危険な脆弱性の一覧が送付される。

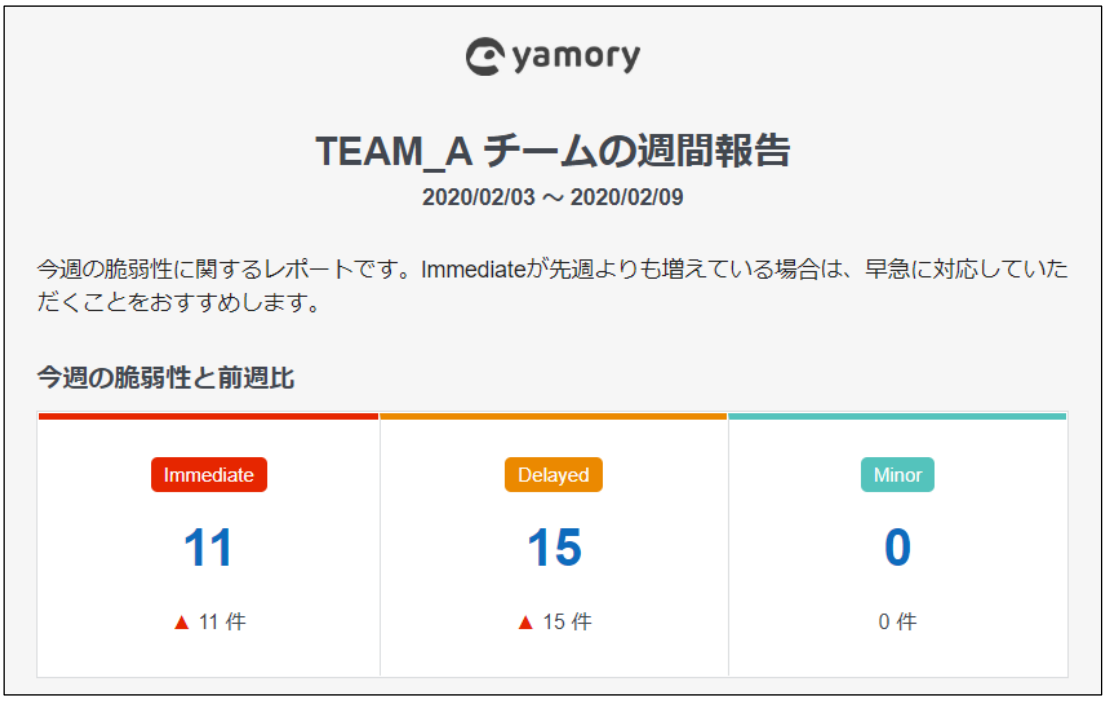

図 21 週間報告(今週の脆弱性と前週比)

図 22 週間報告(検出された脆弱性)

週間報告が届くことで、脆弱性対応の推移を把握できる。

5.6. 導入容易性

5.6.1. 検証項目 G6-1:

利用している OSS 情報を新たに作成することなく利用可能か

検証結果:

ビルドシステム/パッケージ管理システムで保持している OSS の構成情報を利用するため、新たに OSS の構成情報等を作成することなく利用が可能である

検証内容詳細:

2020 年 2 月現在、yamory が対応している開発言語、およびビルドシステム/パッケージ管理システムとして、以下が挙げられている。スキャンの際には、ビルドシステム/パッケージ管理システムで保持している OSS の構成情報を利用するため、yamory 導入時に、OSS の構成情報等を手作業で新規作成する必要はない。

表 2 対応している言語、およびビルドシステム/パッケージ管理システム

開発言語	ビルドシステム/パッケージ管理システム
Java[14]	Gradle, Maven[15]
Ruby[16]	Bundler[17]
Scala[18]	Sbt[19]
PHP[20]	Composer[21]
Python[22]	Pip[23]

[14] Java:プログラミング言語の Java と、Java プログラムの実行環境と開発環境を合わせた演算プラットフォームの双方を指す総称。

[15] Maven:Java 用プロジェクト管理ツール。http://maven.apache.org/

[16] Ruby:国産のオブジェクト指向スクリプト言語。

[17] Bundler:ruby で使用する gem のパッケージ管理システム。https://bundler.io/

[18] Scala:プログラミング言語であり、JAVA 仮想マシン上で動作する。

[19] Sbt:Scala、Java 用のオープンソースのビルドシステム。

[20] PHP:オープンソースの汎用スクリプト言語。

[21] Composer:PHP で使用するパッケージ管理システム。https://getcomposer.org/

[22] Python:汎用のプログラミング言語のひとつ。

[23] Pip:Python のパッケージ管理システム。https://www.pypa.io/

開発言語	ビルドシステム/パッケージ管理システム
JavaScript	npm[24], Yarn[25]
C#(.NET)[26]	NuGet[27]

5.6.2. 検証項目 G6-2:

リモートリポジトリ・ローカルリポジトリのスキャンに対応可能か

検証結果:

GitHub 上のリポジトリ、およびローカルリポジトリにスキャンを実行可能である

検証内容詳細:

2020 年 2 月現在、yamory が対応しているリモートリポジトリは GitHub である。[28]

GitHub リポジトリの初回スキャン時に、GitHub 側の操作で yamory との連携を許可した後は、追加するリポジトリを選択すればスキャンが行われる。

[24] npm:Node.js のパッケージ管理システム。https://www.npmjs.com/

[25] Yarn:JavaScript のパッケージ管理システム。https://yarnpkg.com/

[26] C#(.NET):Microsoft 社が Windows 向けに作成した.NET Framework という開発プラットフォーム上で動作するプログラミング言語。

[27] NuGet:.NET Framework のパッケージ管理システム。https://www.nuget.org/

[28] オンプレミス版の GitHub には未対応。

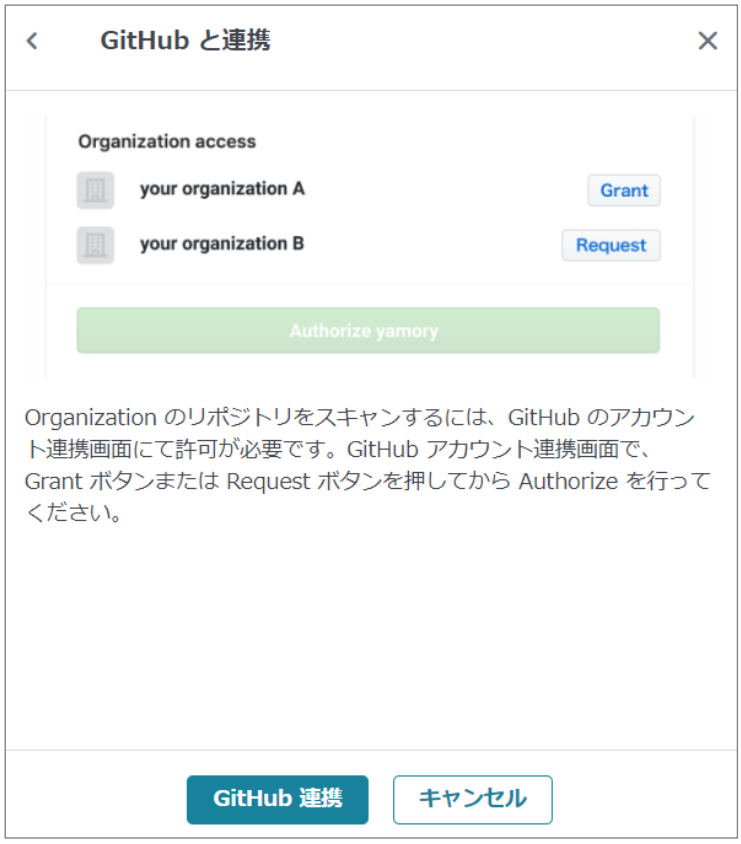

図 23 GitHub との連携画面

なお、yamory では、Java、Scala 等のコンパイル系言語については、間接的に依存しているオープンソースの脆弱性をスキャンするために、GitHub リポジトリのスキャンではなく、コマンドラインからスキャンすることを推奨している。

ローカルリポジトリのスキャンを行う場合には、使用中のビルドシステム/パッケージ管理システムに応じたスキャンコマンドを yamory 上で確認し、ローカル環境内の対象のシステムが含まれるルートディレクトリでコマンドを実行すればスキャンできる。

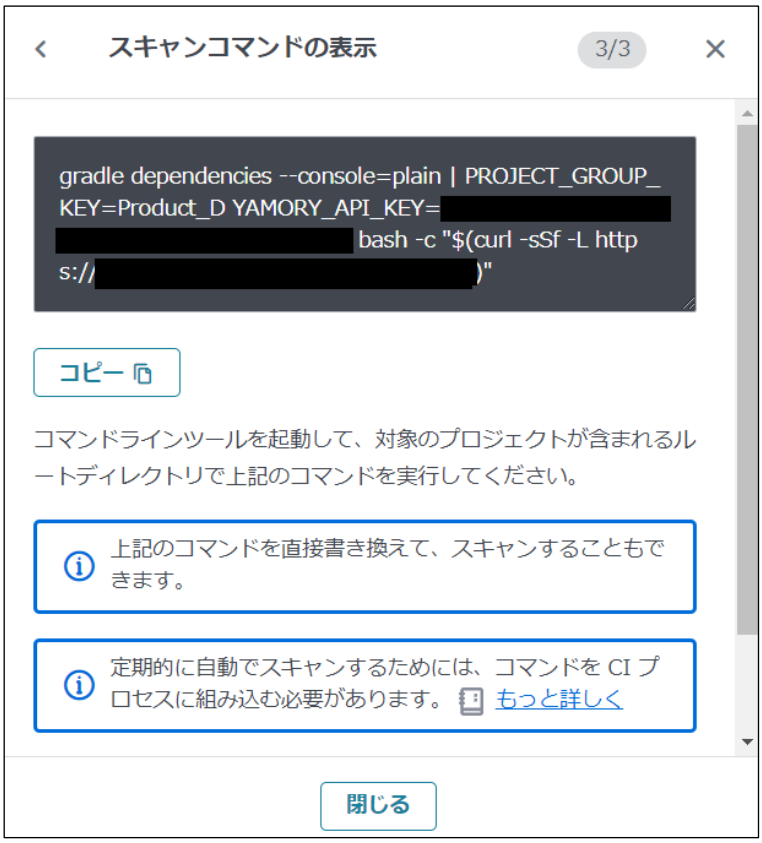

図 24 スキャンコマンドの表示

なお、コマンド実行によるスキャンは、実行環境に bash[29]、および curl[30]がインストールされている必要がある。

5.6.3. 検証項目 G6-3:

エージェントをインストールすることなく利用可能か

検証結果:

エージェントレスであるため、既存の環境に影響を及ぼすことなく、アカウントを登録後、すぐにスキャンが可能である

検証内容詳細:

yamory は SaaS であり、エージェントレスで利用できる。

[29] bash:多くの UNIX/Linux 系 OS で標準的に使われるシェルプログラムのひとつ。

[30] curl:URL を用いてデータ転送を行うコマンドラインツール。https://curl.haxx.se/

リモートリポジトリ（GitHub）では、GitHub アカウントを連携することで、すぐにスキャンが可能である。

ローカルリポジトリに対しても、yamory 上でコマンドを発行し、ローカル環境内の対象のシステムが含まれるルートディレクトリでコマンドを実行すれば、エージェントをインストールすることなく、すぐにスキャンを実行できる。

6. まとめ

yamory は、利用している OSS を自動抽出し、脆弱性を検出、管理するサービスである。

昨今の IT/Web システムの開発においては、OSS の利用が主流となっている。今後も OSS の利用が増加すると考えられ、OSS の管理や、脆弱性への対策がより重要になると言える。

yamory は脆弱性を検出するだけでなく、脆弱性の対応の優先度を示すオートトリアージ機能や、OSS の管理機能も提供する。運用負荷を軽減するための自動的なトリアージ機能により優先度の高い脆弱性から段階的な対応ができる。オートトリアージ機能により、優先的に対応すべき脆弱性が分かるため、セキュリティに対する高度な知識がなくても脆弱性の対処を進めることができ、CSIRT 等のセキュリティ管理チームが存在しない組織であっても扱いやすいツールであると言える。

セキュリティ管理チーム向けの機能では、組織全体の脆弱性の対応状況を俯瞰して、対応指示ができるため、管理タスクを低減ならびに組織のリスク低減に貢献できる。

また、組織の OSS 一覧を管理できているため、ニュースなどで緊急度の高い脆弱性情報が公表された際に、該当するソフトウェアを検索することで、自組織への影響の有無を調査する等、早期の対応ができる点が魅力である。

OSS の管理、脆弱性情報の収集を、人手で行う場合は、運用負荷の高い作業となるが、それらを一元的に yamory に任せることができ、管理が行き届かない昨今の OSS 利用上の課題を解決するツールであると言える。

7. 用語集・略語集

Apache：

Apache HTTP Server の通称であり、Web アプリケーションサーバ。

https://httpd.apache.org/

bash：

多くの UNIX/Linux 系 OS で標準的に使われるシェルプログラムのひとつ。

Bundler：

ruby で使用する gem のパッケージ管理システム

https://bundler.io/

C#(.NET) ：

Microsoft 社が Windows 向けに作成した.NET Framework という開発プラットフォーム上で動作するプログラミング言語。

CI：

継続的インテグレーション（Continuous Integration）。ソフトウェア開発において、ビルドとテストを頻繁に繰り返し、問題を早期に発見して開発の効率化を目指すための手法。

Composer：

PHP で使用するパッケージ管理システム。

https://getcomposer.org/

CPE：

共通プラットフォーム一覧(Common Platform Enumeration)。情報システムを構成する、ハードウェア、ソフトウェアなどを識別するための共通の名称基準。

https://www.ipa.go.jp/security/vuln/CPE.html

CSIRT：

Computer Security Incident Response Team。コンピュータ、またはネットワークで発生した主にセキュリティ上の問題を監視するとともに、問題が発生した場合の調査/分析を行う組織。

curl：

URL を用いてデータ転送を行うコマンドラインツール。

https://curl.haxx.se/

CVE (CVE-ID) :

共通脆弱性識別子 CVE(Common Vulnerabilities and Exposures)は、個別製品中の脆弱性を対象として、米国政府の支援を受けた非営利団体の MITRE 社が採番している識別子。

https://www.ipa.go.jp/security/vuln/CVE.html

CVSS:

共通脆弱性評価システム CVSS（ Common Vulnerability Scoring System）は、情報システムの脆弱性に対するオープンで汎用的な評価手法のひとつで、基本評価基準、現状評価基準、環境評価基準の 3 つ指標を用いて、0.0～10.0 のスコアで評価する。

https://www.ipa.go.jp/security/vuln/CVSS.html

GitHub:

ソフトウェア開発のプラットフォーム。

https://github.co.jp/

Gradle:

オープンソースのビルド自動化システム。

https://gradle.org/

Java:

プログラミング言語の Java と、Java プログラムの実行環境と開発環境を合わせた演算プラットフォームの双方を指す総称。

JavaScript:

主に Web ページで複雑な機能を実行するためのプログラミング言語のひとつ。

Jenkins:

オープンソースの継続的インテグレーションツール。

https://jenkins.io/

JVN:

JPCERT/CC と情報処理推進機構（IPA）が共同で管理している脆弱性情報データベースである。

https://jvn.jp/

Linux：

オープンソースのオペレーティングシステム。

Maven：

Java 用プロジェクト管理ツール。

http://maven.apache.org/

Npm：

Node.js のパッケージ管理システム。

https://www.npmjs.com/

NuGet：

.NET Framework のパッケージ管理システム。

https://www.nuget.org/

NVD：

National Vulnerability Database。アメリカ国立標準技術研究所（NIST）が運用する脆弱性データベース。

https://nvd.nist.gov/

OSS：

Open Source Software の略称であり、人間が読み取れるソースコードを使用、研究、再利用、修正、拡張、そのソフトウェアのユーザーによる再配布可能なソフトウェアを指す。

PHP：

オープンソースの汎用スクリプト言語。

Pip：

Python のパッケージ管理システム。

https://www.pypa.io/

PoC：

Proof of Concept（概念実証）。セキュリティの分野では概念実証コード等とも呼ばれ、脆弱性を悪用した攻撃が実際に有効であることを示すプログラムを指す。

Python：

汎用のプログラミング言語のひとつ。

Ruby：

国産のオブジェクト指向スクリプト言語。

SaaS：

Software as a Service。ソフトウェアを利用者側に導入せず、提供側のサーバで稼働しているソフトウェアをネットワーク経由で利用するサービス。

Sbt：

Scala、Java 用のオープンソースのビルドシステム。

Scala：

プログラミング言語であり、JAVA 仮想マシン上で動作する。

Slack：

ビジネスチャットツール。

https://slack.com/

Struts2：

オープンソースの Java Web アプリケーションフレームワーク。

https://struts.apache.org/

Ubuntu：

Debian GNU/Linux をベースとしたオペレーティングシステム。

UI：

ユーザーインターフェース。ユーザーとコンピュータ、ソフトウェアで情報をやり取りする仕組み。

Web アプリケーション：

インターネット（ウェブ）などのネットワークから利用するアプリケーションソフトウェア。Web サーバ上で動作し、Web ブラウザを使用して利用する。

Windows：

アメリカ Microsoft 社が開発、販売するオペレーティングシステム。

Yarn：

JavaScript のパッケージ管理システム。

https://yarnpkg.com/

アプリケーションレイヤ：

システムを構成する階層のひとつ。システムは大まかにハードウェア、OS、ミドルウェア、アプリケーションの 4 つの階層に分けられる。

アンチウイルス：

コンピュータウイルスを検出・除去するためのソフトウェア。ウイルスなどの特徴を記録したデータファイルとコンピュータ内部でやり取りされるデータを比較し、ウイルスなどを検出する「パターンマッチング型」と、検査対象のデータを自動的に解析し、ウイルスのような不審な振る舞いをするプログラムコードやウイルス特有のプログラムコードが含まれていれば、ウイルスとして検出する「ふるまい検知型」がある。

インシデント：

ISO22300 (2.1.15)では「中断・阻害、損失、緊急事態、危機に、なり得るまたはそれらを引き起こし得る状況」と定義されている。

インタフェース：

異なる 2 つのものを仲介するという意味を持ち、コンピュータ機器同士や、コンピュータとプログラムを結ぶ共用部分、ハードウェア同士を繋げるコネクタ部分等を指す。

エージェント：

IT の分野では、ユーザーや他のシステムの代理としての機能や、複数の要素間で仲介役として機能するソフトウェアやシステムなどを指す。

オンプレミス：

サーバやネットワーク、ソフトウェア等の情報システムを自社の中で保有し、自社内の設備によって運用すること。

クラウド：

インターネット等のネットワーク経由で、ユーザーにサービスを提供する形態。

コマンドライン：

キーボードで入力したテキストで命令を出してコンピュータと直接対話するように操作を行う方法。

ジョブ：

コンピュータが処理する仕事の単位。

ダッシュボード：

様々なデータを収集して簡潔にまとめ、集約して表示する画面を指す。

トリアージ：

セキュリティにおけるトリアージとは、発生した事象の対応可否や緊急度等を勘案し、対応の優先順位を決めることを指す。

パッケージ管理システム：

プログラムの実行ファイルや設定ファイル、ライブラリ等を管理するツールやシステム。

ビルドシステム：

プログラムのソースコードから、実際に動くプログラムを作成するツールやシステム。

ミドルウェア：

オペレーティングシステムとアプリケーションの中間に位置し、特定の機能に特化してオペレーティングシステムとアプリケーションを補佐するソフトウェア。

リポジトリ：

アプリケーション開発時に、システムを構成するデータやプログラム情報を格納したデータベースを指す。インターネット上のクラウド等にリポジトリが置かれている場合にはリモートリポジトリ、ローカル環境の場合はローカルリポジトリと呼ぶ。

構成情報：

ハードウェアやソフトウェアの名称、バージョン情報、ネットワーク機器の IP アドレス等、IT システムを構成する様々な情報。

脆弱性：

ソフトウェア等におけるセキュリティ上の弱点。

脆弱性データベース:

脆弱性情報をデータベース化し、一般向けに公開されているもの。

以上