環境情報を用いた脆弱性検出のチャットボット

アーノ 有里紗[1]　石田 愛[1]　工藤 瑠璃子[2]　渡邊 裕治[1]

概要：日々報告される脆弱性に対して、システム管理者は迅速に対応する必要がある。ChatBot は刻々と変わるシステム状態に応じてタイムリーな通知が可能なため、システム管理者が行う定型業務に対する効率的なインタフェースを提供する。しかしながら，脆弱性対応に ChatBot を活用する場合，日々発見される脆弱性およびその対応を必要とするシステム数が多く，また脆弱性対応プロセスが多様である，といった課題がある．
本論文では，Chatbot フレームワーク「SecBot」を提案する．SecBot は，脆弱性毎のリスクを動的に分析し，最新の脆弱性情報とシステムの稼働状況の知識を獲得し，実運用環境に即した対応の優先度の高いシステムを抽出することにより，人間が効率的に Chat インターフェースから処理できるようにする，という特徴を持つ．実装の結果，対応処理開始までの時間は 43%まで短縮でき，優先度の高いタスクの絞り込みによりメッセージのサイズは 40%削減可能となる．

キーワード：チャットボット，コンテナセキュリティ，リスクスコア

ChatBot System for Vulnerability Detecting using Environmental Metrics

ALISA ARNO[1]　AI ISHIDA[1]　RURIKO KUDO[2]　YUJI WATANABE[1]

Abstract: Chatbot is a widely accepted approach to improve operational efficiency in system management activities. But its application to vulnerability management requires it to address unique challenges such as multiple vulnerabilities, managed systems and highly complex processes of vulnerability patch and remediation.
We propose a new chatbot framework ”SecBot”, which allows operators to focus on priority tasks with finer risk assessment and user friendly interaction via chat-bot interface. We designed risk aggregation strategy to find the highest risk vulnerabilities and systems allowing to focus on higher priority tasks. By using SecBot, the time before initiating the tasks and the message size for interaction is reduced. These evaluations shows that our approach works well to improve vulnerability management tasks.

Keywords: chatbot, container security, risk score

1. はじめに

日々膨大な脆弱性が報告され，システム管理者は新しい脆弱性に対して迅速に対応することが求められるが，新たな脆弱性が公開されてからシステムに対応する脆弱性が検知されるまで時間が空いてしまう．そのため多くの組織では迅速に脆弱性を検知，対応しなければならない課題に直面している [5] [15] [11].

システム管理の効率化のためにチャットボットが活用され始めた．チャットボットは刻々と変化するシステム状態に応じてリアルタイムな通知を可能にする [2] [10]．一方で，脆弱性対応業務に活用する場合，(1) 脆弱性の数，(2) 脆弱性の対応を必要とするシステム数，(3) 多様な脆弱性対応プロセス，という課題がある．

本論文では，上記の課題を解決するチャットボットフレームワーク「SecBot」を提案する．図 1 に SecBot のイメージを示す. SecBot は以下のような特徴を有する．

- 脆弱性対応プロセスをチャットインターフェースから

[1] 日本アイ・ビー・エム 東京基礎研究所
IBM Research
[2] お茶の水女子大学
Department of Information Sciences Ochanomizu University

図 1　SecBot のイメージ図

処理可能にする．

- 脆弱性毎のリスクを分析し，最新の脆弱性情報に基づき，対応の優先度が高いシステムを抽出する．
- システムの稼働環境を検査し，その環境に応じたリスクの重み付けを行うことによって，より実運用環境にそくしたリスクを提示する．

本論文の貢献は，1) チャットインターフェースを活用した脆弱性対応業務の支援，2) 脆弱性リスクとシステム稼働環境の分析による優先タスクの絞り込み手法の提案である．SecBot の脆弱性対応業務効率化における有効性を評価するために，脆弱性検出ツールおよび会話管理 API [22] を用いて SecBot のプロトタイプ実装を行った．実験の結果，SecBot を用いることで，SecBot を用いない場合に比べ対応処理開始までの時間は平均 12.26 秒 (43.78 %) 短縮化することが可能となった．また脆弱性のリスク分析によって優先度の高いタスクを絞り込むことにより，絞り込む前に比べ，表示されるメッセージサイズは 60 %に削減された．これにより，脆弱性対応業務を効率化するためにリスク分析機能を有する SecBot が有効であることがわかった．

本論文の構成は次のとおりである．２章で関連研究を述べ，３章で提案手法である SecBot の構成について述べる．４章ではリスク分析による優先度自動付与手法について示す．５章で提案手法の実装による評価結果を示すとともに有効性について議論し，６章でまとめを示す．

2.　関連研究

2.1　チャットボット

チャットボットとは，人工知能を用いて人間の会話を作り出すコンピュータプログラムである．最初のチャットボットは 1966 年，ドイツ人の計算機科学者でもあり，マサチューセッツ工科大学の教授でもあるジョセフ・ワイゼンバウムの手によって作られた．ELIZA [23] と名付けられたチャットボットは自然言語処理コンピュータプログラムで，ワインゼンバウムは人間と機械が会話をできるという可能性を示すために ELIZE を作り出した．ELIZA は人間が入力したキーワードやフレーズを用いて，事前にプログラムされている回答例から回答を作る．その後，ELIZA を元にして作られた PARRY [16] が，アメリカ人の精神科医であるケネス・コルビーの手によって 1972 年に登場した．ELIZA はシンプルなプログラムであったため，会話をするには限りがあった．PARRY の方が技術的に発展しており，妄想型統合失調症の挙動を示すようにプログラムされている．

ELIZA や PARRY の他にも，1983 年にウィリアム・チェンバレンとトーマス・エルターが RACTER [4] を，1995 年にリチャード・ウォーレスが A.L.I.C.E. (Artificial Linguistic Internet Computer Entity) [21] を作り出した．しかしこれら４つは入力された単語からすでにプログラムされてある用意された回答をただ返す，シンプルな会話ボットである．近年，チャットボットはただの会話ボットだけではなく，ユーザビリティの向上，業務改善や効率化を行うために用いられるようになった．例えば Pribots [12] は，ユーザがチャットボットを使ってアプリケーションのプライバシの設定を行うことを提案している．新しくアプリケーションを使い始めたユーザは，設定画面にたどり着くまでに何ステップも必要であったり，アプリケーションによってプライバシ内容が異なる．Pribots はボットに単語を投げかけるだけで，今のプライバシ状態の確認やセッティングを行うことができる．

他にも銀行や保険会社などの様々な会社でチャットボットがお客様窓口の一サービスとして提供されている [19]．ユーザは聞きたいことや知りたいことのキーワードを打ち込むと，チャットボットが関連する質問を提示したり，答えとなるページを返答する．ユーザはオペレータにつながるまでの長い時間を待たずにすみ，気軽に質問できるサービスを提供することによって，顧客満足度の向上と業務改善が可能である．さらにセキュリティにも応用することが可能であり，Lenny [17] はスパムコールに対して返答を行うフォンボットであり，スパムコールと Lenny が十数分にも会話し，電話を終わらせる．

2.2　コンテナの脆弱性判断

本論文で提案する SecBot はコンテナイメージの脆弱性情報を用いる．近年コンテナの脆弱性を評価する様々なツールや手法が提案されている．Docker's Benchmark for Security [9] は Docker Inc. から提供されているツールで，コンテナのインスタンス内でスクリプトを実行することによってセキュリティスキャンを行う．Docker Security Scanning [8] は既知の脆弱性によって影響を受ける可能性があるコンポーネントをイメージ内から見つけ出す．Banyan Collector [3] はコンテナやイメージにインストールされたパッケージ情報を収集するフレームワークである．Clair [6] はコンテナに脆弱性がないかどうかを静的にスキャンするオープンソースである．OpenSCAP Container Compliance [14] は稼働しているコンテナやイメージに対して脆弱性についての CVEID を提供する．Twistlock [20]

はコンテナ環境を守るセキュリティツールであり，自身の環境にあるコンテナイメージに脆弱性がないかどうか調査し，また稼働している環境に対して挙動を分析・監視を行う．Amazon Inspector [1] は AWS にデプロイされたアプリケーションのセキュリティとコンプライアンスを向上させるための、自動化されたセキュリティ評価サービスである．IBM Vulnerability Advisor [13] は自動的にコンテナイメージをスキャンし，ユーザに脆弱性情報を送る．Rui Shu らは Docker Hub の公式およびコミュニティのイメージの発見・ダウンロード・分析を自動化するフレームワークである Docker image vulnerability analysis (DIVA) [18] を提案した．

脆弱性の深刻度は共通脆弱性評価システム (CVSS) [7] から計算される．CVSS では基本評価基準，現状評価基準そして環境評価基準の 3 つの基準で脆弱性を評価する．基本評価基準および現状評価基準は脆弱性情報を発表する組織によって評価および算出され，環境評価基準はユーザの環境においてその脆弱性がどれだけ脅威となるかを表す．上記の関連研究では主に基本評価基準を用いている．しかしながら，脆弱性の深刻度はユーザの計算機環境によって変化するため，基本評価基準のみを用いるのでは正しい結果を計算することができない．そこで本論文ではより精密な深刻度を計算するために環境評価基準を自動的に計算する．

3. SecBot

コンテナ利用の増加により，コンテナ管理者は全てのセキュリティを把握することが困難になっている．コンテナ管理者が行うべき脆弱性対応作業には以下のようなものが挙げられる．

(1) コンテナを管理するページを開く
(2) ユーザ情報とパスワードの入力
(3) 承認されるのを待つ
(4) 管理ページにログインする
(5) コンテナのリストを表示される
(6) コンテナ 1 に脆弱性がないかどうか調べる
(7) 結果を待つ
(8) コンテナ 1 の脆弱性に対応する
(9) コンテナ 2 に脆弱性がないかどうか調べる
(10)結果を待つ
(11)コンテナ 2 の脆弱性に対応する
(12)全てのコンテナの脆弱性を調べ終わるまでステップ 6 から 8 を繰り返す

従来はコンテナ管理者がホストにログインし個別に対応したり，スクリプトをサーバから実行し対応を行ってきた．しかしながら，従来法では管理するシステムや脆弱性が膨大に増えた場合や，リアルタイムでの脆弱性対応に限りがある．そこで本論文では，チャットボットの手法を活用し，脆弱性対応を行うフレームワークである SecBot を提案する．SecBot は上記の脆弱性対応作業をホストログインやスクリプトによらず，チャットインターフェースで実行可能にする．SecBot を使うことによって，コンテナの脆弱性がより管理しやすくなり，脆弱性対応業務の効率化が可能となる．

コンテナ管理者は以下の作業をするだけで良くなる．

- 脆弱性が発見されたかどうか通知を受ける．
- コンテナのリストを表示する．
- 管理しているコンテナに脆弱性があるかどうか調べる．
- 脆弱性に対応する．

例えば，SecBot は管理者に脆弱性発見通知を出し，管理者は SecBot に脆弱性の詳細な情報を尋ねる．SecBot が結果を表示し，管理者は脆弱性を無くすために必要な作業を行う．表 1 に SecBot との会話の一例を示す．

表 1 SecBot との会話の一例

SecBot:	脆弱性が検知されました．
Admin:	詳細情報を表示してください．
SecBot:	10 個のコンテナに 100 個の脆弱性が発生しています．
Admin:	全てのコンテナと脆弱性を見せてください．
SecBot:	コンテナは .....
	脆弱性は .....
Admin:	.....

3.1 SecBot の構成図

図 2 に SecBot の構成を示す．ユーザはもうすでにチャットインターフェースにログインしているため，再びコンテナ管理ページにログインする必要はない．以下に SecBot の仕組みを示す．

a.1 新しいコンテナがプッシュされるたびに，データ収集ツールがコンテナの情報をクロールする．
a.2 クロールした情報を定期的に脆弱性検出モジュールに送る．
b 脆弱性検出モジュールが脆弱性を検出した場合，チャットインターフェースに通知を送る．
c チャットインターフェースがユーザに通知を出す．
d ユーザは表 1 に示したようなやり取りを行う．
e ユーザから入力されたメッセージをもとに，必要な API を呼ぶ．
f API が情報を返す．
g チャットインターフェースが情報をユーザに表示する．

次章で SecBot のシステムモデルを示す．

3.2 システムモデル

表 2 に SecBot のシステムモデルを示す．本論文ではチャットインターフェースに Watcon Conversation および Hubot を，脆弱性検出モジュールに Vulnerability Detection Tool を，データ収集に Agentless System Crawler を用いた．Watson Conversation は日本アイ・ビー・エムが提供してい

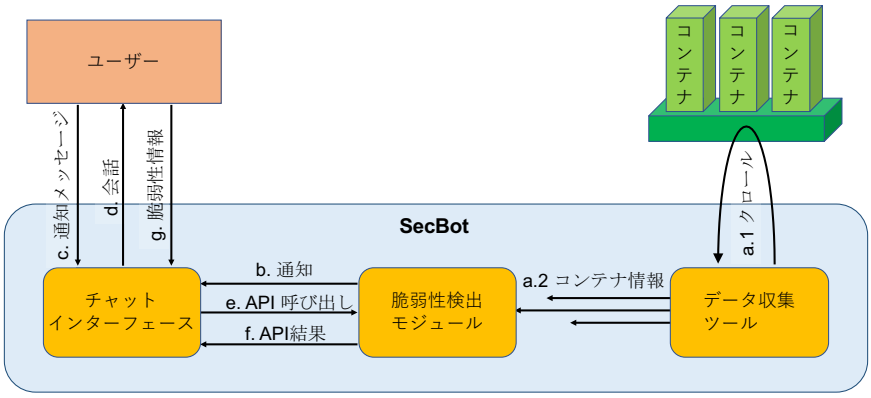

図 2　SecBot の構成図.

表 2　システムモデル

チャットボットインターフェース	Watson Conversation & Hubot
脆弱性検出モジュール	Vulnerability Detection Tool
データ収集	Agentless System Crawler

るチャットボットのような会話アプリケーションである．Hubot は GitHub ops．が提供しているチャットボットシステムである．Vulnerability Detection Tool がコンテナのセキュリティを調べ，Agentless System Crawler はクラウドプラットフォームを観察，分析し，データを収集するフレームワークである．

図 3 に SecBot のアルゴリズムを示す．Vulnerability Detecting Tool は Agentless System Crawler を用いて常にコンテナの脆弱性を調べる．脆弱性が検出されたなどの事象が発生した場合，Vulnerability Detection Tool は Watson Conversation に通知を送る．その後 Watson Conversation は Hubot に通知を送り，Hubot はユーザにアラートを出す．ユーザはそのアラートに対処するために，必要な情報を Bot に尋ねる．Bot はユーザが求めている情報を特定の API から取得し，ユーザに返す．ユーザは得られた情報から脆弱性を無くすために必要な作業を行う．

しかし，Vulnerability Detection Tool をそのまま用いるには限界がある．コンテナや脆弱性の数が大きくなると情報量が膨大になり，チャットボットの画面で重要な脆弱性を追うのが困難になってくる．そこで脆弱性に優先度を付けることによって効率的に脆弱性に対応できるようにする．

4. リスク分析エンジン

3.2 章で述べたようにチャットボットを用いてコンテナの脆弱性情報を提示するには情報量が多くなりすぎる．本章ではそれらの問題を解決するために CVSS(Common Vulnerability Scoring System) を用いて各脆弱性の重要度を算出し，さらにそれを用いて各コンテナのセキュリティリスクを算出するリスク分析エンジンを提案する．Secbot に提示する情報をより重要なものに限定することが可能となる．

4.1 全体の構成図

図 4 にリスク分析エンジンの構成図を示す．それぞれの環境で正確なリスクスコアを算出するために，リスク分析エンジンでは CVSS，コンテナクラウド上で取得できるコンテナの環境情報とそこから導出されるパッケージ脆弱性，そして脆弱性情報データベースを用いる．以下にリスク分析エンジンで行われるプロセスを示す．

(1) ユーザはコンテナクラウド上にホストされたコンテナ ID をリスク分析エンジンに入力する．
(2) リスク分析エンジンはコンテナ ID を入力し，コンテナクラウド上の環境情報取得 API を呼び出す．
(3) リスク分析エンジンはコンテナの環境情報 (インストールされているパーケージ一覧，SSH サーバの有無，リモートログイン設定等) を取得する．
(4) リスク分析エンジンは環境情報を用いて vulnerability detection module を呼び出す．
(5) Vulnerability detection module はパッケージ脆弱性を検出し，図 5 のような CVEID と共にリスク分析エンジンに返す．
(6) リスク分析エンジンは CVEID をキーとして脆弱性情報データベースを入手する．
(7) リスク分析エンジンは CVSS 情報を取得する．
(8) リスク分析エンジンは CVSS 情報と環境情報を元に環境スコアを算出する．

本論文では脆弱性情報データベースとして XForce Exchange API [24] を，vulnerability detection module として既存の脆弱性検出エンジンを用いる．

4.2 環境情報を用いた検出精度の向上のためのアプローチ

パッケージ脆弱性の情報は CVSS の形で利用可能である．一方で，パッケージ脆弱性の情報は脆弱性そのものに対する評価であり，そのパッケージを実際に利用する個別のシステムの構成や周辺環境は考慮されていない．例えば，root ログインパスワードを不正取得できる脆弱性があったとする．このとき仮にシステムがパスワードによるログイ

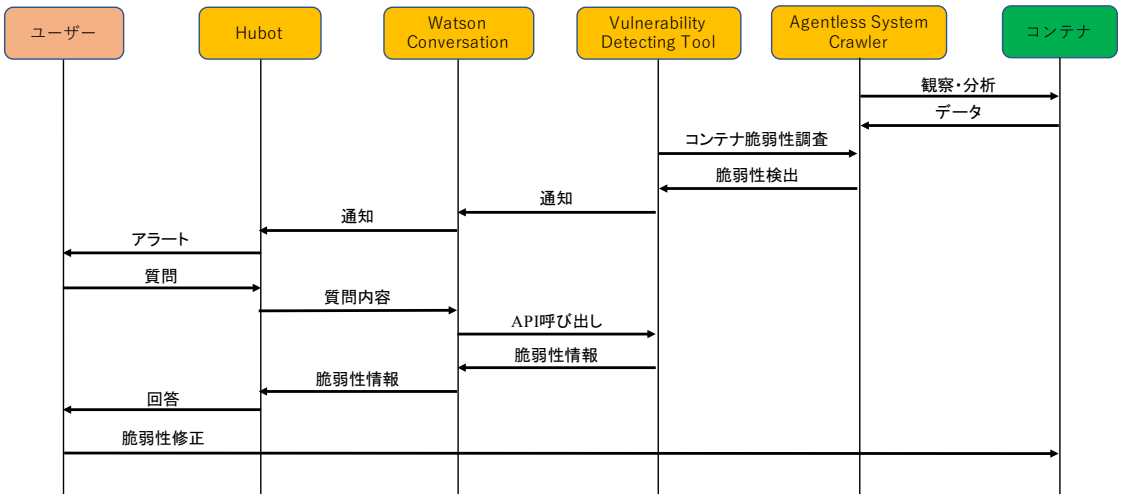

図 3　SecBot の実装

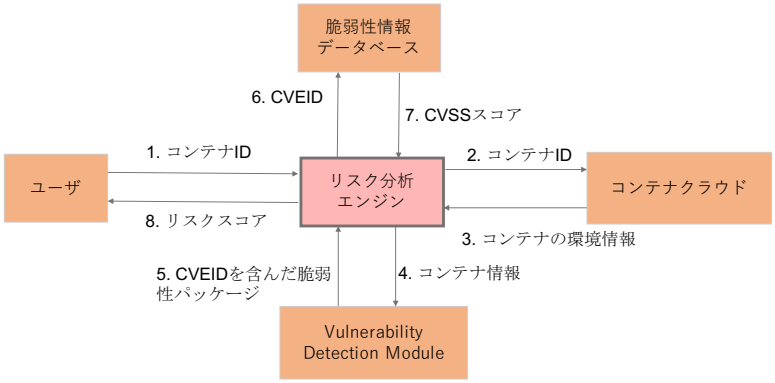

図 4　リスク分析エンジンの構成図.

```
{
  "DSA-3960-1": [ "CVE-2017-7526" ],
  "DSA-3982-1": [ "CVE-2017-12837", "CVE-2017-12883" ]
}
```

図 5　脆弱性情報および CVEID の一例.

ンが無効になるように設定されているならば，そう設定されていないシステムに比べ，ログインパスワード漏洩のリスクは異なるものになるのは明らかである．つまり，あるパッケージ脆弱性があっても，特定のシステムの構成によりその脆弱性をついた攻撃が有効になるような状況が生じないのであれば，攻撃成功のためのハードルが上がるという意味で，その脆弱性に対するリスクは低くなる，と考えられる．このように，各個別の具体的なシステムの構成や実際の周辺環境に関する情報を本稿では「環境情報」と呼ぶ．環境情報を利用することによって，個別のシステムの状況を考慮して各リスク要素の重み付けを考慮した CVSS スコアを計算することができる．このスコアは CVSS において環境スコア (Environmental Score) と呼ばれる．この環境スコアにより，脆弱性情報は対象システムや対象の組織の状況に基づいた評価が行われるため，組織・システムの脆弱性をより正確に把握し対策を講じることが可能となる．本論文では，この環境スコアの導出を自動化するためのアプローチを提案する．自動化のステップの概略を以下に示す．

(1) 自動化モニタリングモジュール (図 6 (1)) がインストールされたパッケージ，アプリケーション，ネットワーク設定などのシステム構成を集める．

(2) Vulnerability detection module (図 6 (2)) がインストールされたパッケージから例えば，脆弱性 CVE-2017-18001 を検出する．この脆弱性は外部の攻撃者がルートアクセス可能であることを示す．

(3) CVEID から基本評価基準を取得する．今回の例においては，CVE-2017-18001 の評価は 9.8 である．

(4) 同時刻に，自動化モニタリングモジュールが集めたシステム構成の情報から環境情報を構成する．システム構成の情報からルートログインを許可していない情報が得られたとする．

(5) 得られた環境情報から環境評価基準を計算する．今回の例では，ルートアクセスを可能とする脆弱性 CVE-2017-18001 の評価 9.8 は，ルートログインを許可していないという環境情報から，評価は 7.5 と再計算される．

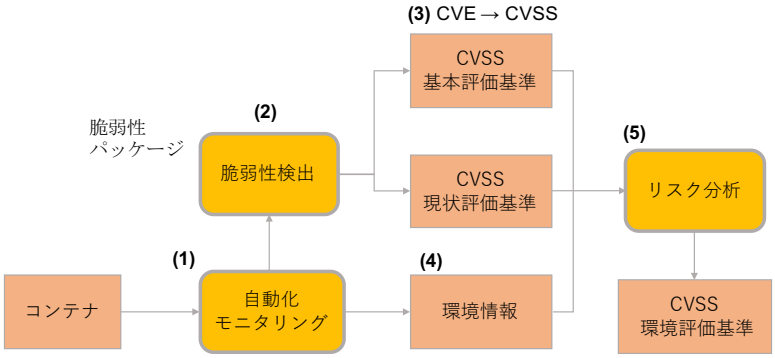

図 6　環境情報を計算するためのステップ

4.3　全体のリスクスコア

各脆弱性のリスクスコア上記に示した通りである．しかしながら，リスクスコアをチャットボットで使用する場合，コンテナの全体のリスクスコアを用いたほうが，コンテナ内のそれぞれの脆弱性のリスクスコアを用いるよりも有用である．そこで本章でコンテナ全体のリスクスコアの算出方法を示す．

ここで，C をコンテナのグループ，$pkg(c) = (p_1, \ldots, p_k) \in P^k$ を各コンテナ $c \in C$ にインストールされたパッケージのリストとする．各パッケージ $p \in pkg(c)$ には脆弱性 $vul(p) = v_1, \ldots, v_l \in V^l$ をもつ．ここで V は既知の脆弱性セットであり，それぞれの脆弱性 $v \in V$ は CVEID で区別できるとする．脆弱性検出関数 F_{VD} は，$c \in C$ において全ての $p \in pkg(c)$ の $F_{VD}(c)$ が $(p, vul(p))$ を持っているときに使用可能であると仮定する．また各脆弱性 $v \in V$ は，CVSS および外部の脆弱性情報 F_{VS} から表される独自の指標 m を持っていると仮定する．

各コンテナが実際に実行されている環境に合わせて調整されたリスク指標を使用する．時間 t での各コンテナ $c \in C$ の環境情報 $e = env(c, t)$ はクローラーによって取得する．章 4.2 で示した通り，環境情報 e は脆弱性 $env(c, t)$ の環境スコアを計算するのに用いる．これらのスコアは各脆弱性によって算出されるため，各パッケージでの脆弱性スコアと，コンテナ内全てのパッケージの脆弱性スコアを求めるロジックが必要である．そこでパッケージレベルとコンテナレベルのリスク評価関数を定義する．最も高いスコアが敵に狙われやすいため，スコア計算には max 関数を使用する．上記のアルゴリズムをアルゴリズム 1 に示す．二つのリスクスコアは max を用いて以下のように計算される．ここで $\hat{r}[p]$ をパッケージ p から影響を受けている脆弱性のスコアのセット，$\bar{r}[c]$ をコンテナ c にインストールされているパッケージから算出されたパッケージレベルのスコアのセットと定義する．

$$agg_p(\hat{r}[p]) = \max(r, \forall(v, r) \in \hat{r}[p])$$

$$agg_c(\bar{r}[c]) = \max(r, \forall(p, r) \in \bar{r}[c])$$

Algorithm 1 全体のリスクスコア

procedure リスク集合
  $c \leftarrow C$
  $t \leftarrow$ クロールした時間
  $e \leftarrow env(c, t)$
  $F_{VD} : C \rightarrow (P \times V^*)^* \leftarrow$ *VD* 関数
  $F_{VS} : V \rightarrow M \leftarrow$ 脆弱性情報
  $risk_e$: 環境情報
  agg_p : パッケージレベルのリスクスコア
  agg_c : コンテナレベルのリスクスコア
main:
  $vd = \{(p, vul(p)) : \forall p \in pkg(c)\} \leftarrow F_{VD}(c)$
  for all $p, vul(p) \in vd$ **do**
    $\hat{r}[p] = \emptyset$
    for all $v \in vul(p)$ **do**
      $m \leftarrow F_{VS}(v)$
      $r \leftarrow risk_e(m, e)$
      $\hat{r}[p] \leftarrow \hat{r}[p] \cup \{(v, r)\}$
    $r_p = agg_p(\hat{r}[p])$
    $\bar{r}[c] \leftarrow \bar{r}[c] \cup \{(p, r_p)\}$
  $r_c \leftarrow agg_c(\bar{r}[c])$ **return** r_c
agg_p:
  $agg_p(\hat{r}[p]) = \max(r, \forall(v, r) \in \hat{r}[p])$
agg_c:
  $agg_c(\bar{r}[c]) = \max(r, \forall(p, r) \in \bar{r}[c])$

このように算出したリスクを用いることで，ユーザは SecBot に提示する脆弱性情報を大幅に削減することが可能となる．

5.　特性評価

提案した手法の効果を評価するため，外部コンポーネントとして脆弱性検知ツールと会話管理 API を用いて [22]SecBot を実装した．Ubuntu，Debian，CentOS などのオープンソースの OS イメージで構築された 15 個のコンテナを用いて評価を行った．まず，コンテナ内のパッケージ一覧と，脆弱性のあるパッケージ，およびその脆弱性に対応する CVEID を含む各コンテナの脆弱性レポートを取得し，4 章で定義したリスク評価関数を用いてリスク指標を計算する．

テストで用いた各コンテナで見つかった実際のパッケー

表 3　評価データ

コンテナ名	c1	c2	c3	c4	c5	c6	c7	c8	c9	c10	c11	c12	c13	c14	c15
脆弱性パッケージの数	8	5	5	5	5	5	1	2	2	1	5	2	1	9	3
パッケージ合計数	191	384	117	114	265	384	385	106	134	101	164	123	112	376	198
脆弱性の数	28	25	21	9	6	25	17	8	8	1	10	3	1	39	10

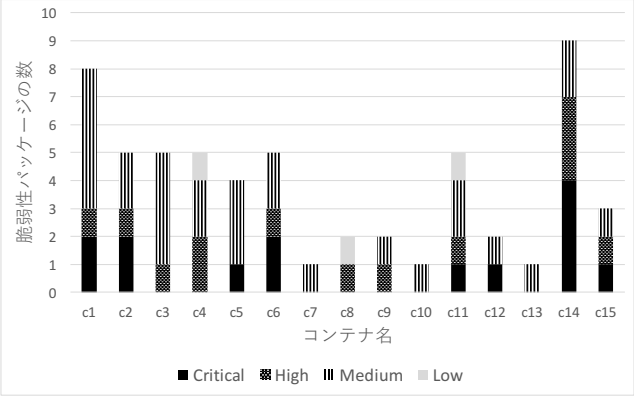

図 7　コンテナに含まれる脆弱性パッケージのリスク分析.

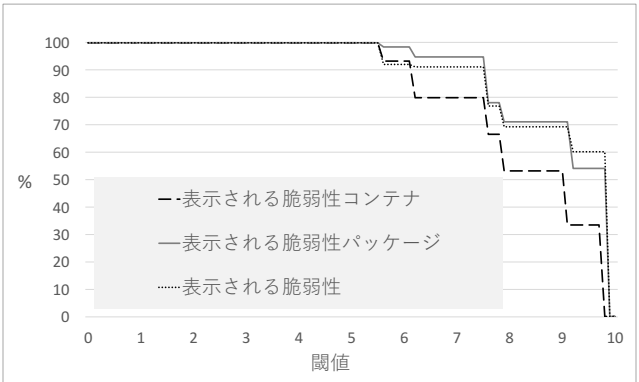

図 8　閾値によって表示される脆弱性コンテナ，
脆弱性パッケージ，脆弱性のパーセンテージの変化.

ジと脆弱性の数を表 3 に示す．すべてのコンテナで脆弱性を含むパッケージがインストールされているが，脆弱性の数とパッケージの数は異なる．各コンテナの脆弱性のあるパッケージの詳細を図 7 に示す．各棒グラフは脆弱性の数と，Critical，High，Medium，Low などの CVSS の深刻度の内訳を表している．例えば，コンテナ c_2 とコンテナ c_4 のどちらにも 5 個の脆弱性が存在するが，コンテナ c_2 には Critical に分類される深刻度をもつ脆弱性がコンテナ c_4 よりも多く含まれている．このため，リスクの重大性の観点から，コンテナ c_2 への迅速な対応が優先されるべき，となる．

5.1　作業時間の比較

表 4　従来方法と，SecBot を用いた時の所要時間の比較

従来方法	SecBot
28.00 秒	15.74 秒

表 4 に従来方法と SecBot を使用した時の，脆弱性を調べるのに要した時間を示す．従来方法では，ユーザがホストページに行き，認証を受け，コンテナの脆弱性が表示されるまでの時間を，SecBot では，SecBot から通知がきた後，脆弱性を見せるように会話する時間を測る．計測回数はそれぞれ 15 回である．測定結果より，従来方法では平均 28 秒，SecBot を用いた方法では平均 15.74 秒であり，平均 12.26 秒（43.78 %）の所要時間削減となった．これは，従来方法では，脆弱性の有る無しにかかわらず，全てのコンテナが表示されるの待ってから，脆弱性の有無を調べるためである．SecBot では，脆弱性のあるコンテナの表示を待つのみなので，短い時間で脆弱性を調べることができる．この結果により，SecBot を用いた方が，より効率的に脆弱性を調べることが可能である．

5.2　チャットメッセージサイズ

本論文のリスク解析を適用する利点の 1 つに，ユーザーと SecBot とのやりとりで発生するメッセージを少なくできるということが挙げられる．これを評価するために，通信の優先度戦略によってどれだけの情報を減らすことができるか見積もる．組織により事前に定義されたリスクスコアの閾値 τ について，リスクスコア s が閾値 τ を超えたときのみ脆弱性の通知を受けるものとする．図 8 は，図 7 で示した各コンテナで見つかった脆弱性のリスク分散スコアから見積もった脆弱性のあるコンテナ，脆弱性のあるパッケージおよび脆弱性の情報のメッセージサイズを示している．

ここで，$\tau = 7$ としたとき，ユーザーと SecBot との会話で発生するメッセージサイズは 20%削減された．より高い閾値を選んだ場合，さらなる削減が期待できる．実際，CVSS v3.0 で Critical な深刻度と定義されている $\tau = 9.7$ を採用した場合，40%以上のメッセージの削減を見積もることができる．これは，全体のリスクスコアがチャットメッセージサイズの削減と，モバイルデバイスなど表示領域が限られた環境におけるチャットボットとの効率的なやりとりを可能としていることが分かる．

5.3　考察

実験から得られた所見を以下に示す．

- メッセージサイズの削減はリスクスコアの閾値に強く依存するため，許容可能なリスクと効率化のバランスで適切な閾値を選ぶことが重要である．図 7 より，多くの脆弱性の深刻度は Critical ではない．そのため Critical の範囲内で閾値を選ぶと SecBot からの通知が

発生しない．しかし SecBot とチャットボットではないインターフェースを組み合わせて使うことで，ユーザーは Critical に比べて緊急度の低い脆弱性に対応できる．

- 多くのコンテナは新しく発見された脆弱性の影響を受ける．図 8 における閾値を挙げたときの急なメッセージサイズの減少は，いくつかの脆弱性が多くのコンテナやパッケージに影響を与えていることを意味している．そのため，それらすべてのコンテナを集約し，新しい脆弱性が見つかったことを通知する 1 つのメッセージを生成することによって，さらなるメッセージサイズの削減が可能となると考えられる．
- 閾値はリスクの大きさによってタスクを優先付けるのにとても直観的な方法であるが，他の手法を我々の優先度手法に取り入れることも可能である．例えば，1 つの脆弱性を修正することにより多くのコンテナが一度に改善される場合，脆弱性の影響を受けるコンテナの数を使うことも考えられる．
- max を用いたリスクスコアの算出は，深刻度が偏ってしまうことがある．実際，コンテナレベルのリスク集約を計算するために単純に max を適用してしまうと，15 個のコンテナのうち 8 個のコンテナの深刻度が Critical になってしまう．個々の脆弱性の情報を利用したパッケージ間のコンテナレベルの集約など，どのような粒度での組み合わせが，このような偏ったリスク評価を避けることに有効かは，今後の課題である

6. 結論

本論文では，脆弱性対応業務の支援および脆弱性リスクとシステム稼動環境分析による優先タスクを絞り込む SecBot を提案した．特性評価により，対応処理開始までの時間は 43%まで短縮でき，優先度の高いタスクの絞り込みによりメッセージのサイズは 40%削減可能となった．よって SecBot は脆弱性管理をより効率化することが有効である．

参考文献

[1] Amazon inspector. https://aws.amazon.com/jp/inspector/.

[2] C. J. Baby, F. A. Khan, and J. N. Swathi. Home automation using iot and a chatbot using natural language processing. In *2017 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–6, April 2017.

[3] Banyan collector. https://github.com/banyanops/collector.

[4] W. Chamberlain. The policeman's beard is half constructed. *Warner Books*, 19(8):4, 1984.

[5] T. C. Chieu, M. Singh, C. Tang, M. Viswanathan, and A. Gupta. Automation system for validation of configuration and security compliance in managed cloud services. In *2012 IEEE Ninth International Conference on e-Business Engineering*, pages 285–291, Sept 2012.

[6] Clair. https://coreos.com/clair/docs/latest/.

[7] Common vulnerability scoring system. https://www.first.org/cvss/.

[8] Docker security scanning. https://docs.docker.com/docker-cloud/builds/image-scan/.

[9] Docker's Benchmark for Security. https://github.com/docker/docker-bench-security.

[10] S. Dutta, G. Joyce, and J. Brewer. Utilizing chatbots to increase the efficacy of information security practitioners. In *International Conference on Applied Human Factors and Ergonomics*, pages 237–243. Springer, 2017.

[11] B. Eshete, A. Villafiorita, and K. Weldemariam. Early detection of security misconfiguration vulnerabilities in web applications. In *2011 Sixth International Conference on Availability, Reliability and Security*, pages 169–174, Aug 2011.

[12] H. Harkous, K. Fawaz, K. G. Shin, and K. Aberer. Pribots: Conversational privacy with chatbots. In *WSF Symposium on Usable Privacy and Security (SOUPS)*, 2016.

[13] IBM vulnerability advisor. https://www.ibm.com/blogs/bluemix/2016/06/docker-container-security-with-vulnerability-advisor/.

[14] OpenSCAP Container Compliance. https://github.com/OpenSCAP/container-compliance.

[15] N. Papanikolaou, S. Pearson, M. C. Mont, and R. K. Ko. A toolkit for automating compliance in cloud computing services. *International Journal of Cloud Computing*, 3(1):45–68, 2014.

[16] R. C. Parkinson, K. M. Colby, and W. S. Faught. Conversational language comprehension using integrated pattern-matching and parsing. *Artificial Intelligence*, 9(2):111–134, 1977.

[17] M. Relieu and A. Francillon. Using chatbots against voice spam: Analyzing lenny's effectiveness. In *Symposium on Usable Privacy and Security (SOUPS)*, 2017.

[18] R. Shu, X. Gu, and W. Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, pages 269–280. ACM, 2017.

[19] Spixii ltd. https://www.spixii.com/.

[20] Twistlock. https://www.twistlock.com/product/vulnerabilitymanagement/.

[21] R. S. Wallace. The anatomy of alice. In *Parsing the Turing Test*, pages 181–210. Springer, 2009.

[22] Watson conversation. https://www.ibm.com/watson/services/conversation/.

[23] J. Weizenbaum. Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[24] Xforce exchange api. https://exchange.xforce.ibmcloud.com/.