

脆弱性対策の効果的な進め方（ツール活用編）

～ 脆弱性検知ツール Vuls を利用した脆弱性対策 ～

目次

はじめに.....	3
本書の対象読者.....	3
1. 昨今の脆弱性を取り巻く状況.....	4
1.1. 脆弱性の公開状況.....	4
1.2. 昨今の脆弱性を悪用した攻撃・被害事例.....	5
1.2.1. Apache Struts2 の脆弱性を悪用した攻撃.....	5
1.2.2. Oracle WebLogic Server の脆弱性を悪用した攻撃.....	5
1.2.3. Drupal の脆弱性を悪用した攻撃.....	6
2. 脆弱性対策の考え方.....	7
2.1. 脆弱性対策のフロー.....	7
2.1.1. 対象ソフトウェアの把握.....	8
2.1.2. 脆弱性関連情報の収集.....	9
2.1.3. 適用の判断.....	10
2.1.4. 計画.....	11
2.1.5. 検証.....	12
2.1.6. 適用.....	13
2.2. 自動化の重要性.....	13
3. 脆弱性検知ツール「Vuls」の紹介.....	14
3.1. 脆弱性検知ツール「Vuls」とは.....	14
3.2. 特徴.....	15
4. Vuls 動作検証.....	17
4.1. 脆弱性検知ツール実行環境およびバージョン.....	17
4.2. 注意事項.....	19
4.3. 動作検証の流れ.....	20
4.4. 事前準備.....	21
4.5. 手動インストール.....	23
4.5.1. go-cve-dictionary をインストールする.....	23
4.5.2. 脆弱性データベースから脆弱性情報を取得する.....	24
4.5.3. goval-dictionary をインストールする.....	25
4.5.4. Redhat から OVAL 情報を取得する.....	26
4.5.5. gost をインストールする.....	27
4.5.6. Redhat から脆弱性情報を取得する.....	28
4.5.7. go-exploitdb をインストールする.....	28
4.5.8. Vuls をインストールする。.....	30
4.6. (参考) Docker インストール.....	32
4.6.1. 前提条件.....	32
4.6.2. セットアップ.....	32

4.6.3. ソフトウェアのバージョン情報を確認する.....	33
4.7. 脆弱性検知.....	34
4.7.1. Local Scan Mode.....	35
4.7.2. Remote Scan Mode.....	37
4.8. レポート確認(Vuls 内機能).....	41
4.9. VulsRepo.....	45
4.9.1. VulsRepo インストール.....	45
4.9.2. VulsRepo 実行.....	46
4.9.3. VulsRepo 利用.....	49
4.10. cron 設定.....	52
5. 検証結果.....	55
おわりに.....	57

はじめに

近年、ウェブサイトの構築に使用される CMS やウェブアプリケーションソフト等の脆弱性を悪用する攻撃が数多く確認されており、それらのソフトウェアを利用している組織にとって重大な脅威となっている。万が一、攻撃者に脆弱性を悪用され、組織内でセキュリティインシデントを発生させてしまった場合、企業機密や顧客情報が漏えいしたり、マルウェアに感染させられたり等、業務に直結するような被害を受けるだけでなく、顧客・取引先の信頼失墜やブランドイメージの低下によるビジネスチャンスの喪失により、組織の事業継続にまで深刻な影響を与える恐れがある。本書では、脆弱性を悪用する攻撃を防ぐために、システム管理者がどのように情報収集を行えば良いのか、また収集した情報をどのように分析して、どう対策に活用すれば良いのか等、脆弱性対策を適切に進めていくための考え方について解説を行う。

また、脆弱性に関する被害状況を受け、脆弱性対策に意識を向けるようになった組織も増えてきている。しかしながら、多くの組織が脆弱性関連情報の収集に関する様々な課題を抱えているのではないかと考える。例えば、収集作業を全て手動で行っており、収集が完了するまでに多くの工数が掛かるケース、管理しているシステムにインストールされたソフトウェアを把握しておらず、適切な脆弱性情報の収集が行えていないケース、急遽システムの管理者に任命されてしまい、収集方法が分からないケース等である。こうした課題がある状態では、脆弱性情報の収集の際に抜け漏れや対応の遅延が発生し、結果として冒頭で述べた深刻な事態に陥る可能性がある。

そこで、本書は脆弱性関連情報の収集における課題を解決する 1 つの案として、Linux/FreeBSD 系サーバにインストールされているソフトウェアの脆弱性の、収集および検知を自動化できる Vuls (Vulnerability Scanner)¹ を活用できないかと動作検証を行った。当該の章では、本ツールの概要と構築手順、および検証結果についてまとめている。

本書が、組織における脆弱性対策の取り組みの推進と、脆弱性対策に対する意識の向上に役立てられることを期待する。

本書の対象読者

・システム管理者

適用例：システムの脆弱性対策を担当されている方

脆弱性対策にツール適用を検討している方

¹ 「脆弱性検知ツール Vuls」 <https://vuls.io/ja/>

1. 昨今の脆弱性を取り巻く状況

1.1. 脆弱性の公開状況

脆弱性情報は日々公表され続けており、IPA が運用している「脆弱性対策情報データベース JVN iPedia」² 日本語版には、2018 年 12 月 31 日までに累計 92,674 件の脆弱性対策情報が登録されている。また、2018 年第 4 四半期（2018 年 10 月 1 日から 12 月 31 日）の 3 か月間に登録された脆弱性対策情報の件数は 3,560 件となっている。これは 1 か月当たり 1,100 件を超える件数が公開されていることになる（図 1-1-1）。³

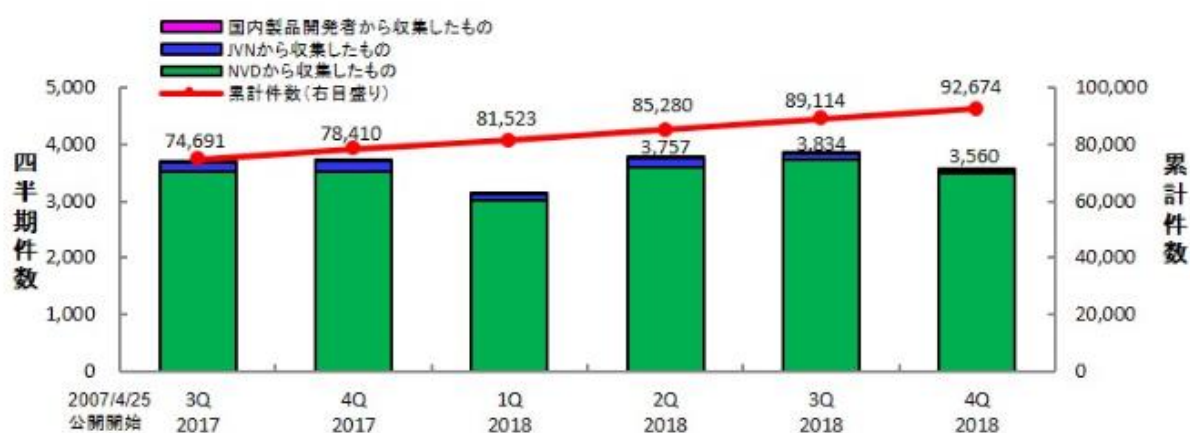


図 1-1-1 JVN iPedia の登録件数の四半期別推移

しかし、登録されている全ての脆弱性がシステムに影響を与える訳ではない。公表される脆弱性の中には、多くの利用者が存在する Apache Struts のようなオープンソースソフトウェアの脆弱性が登録されることもあれば、用途が限定的で利用者が少ないソフトウェアの脆弱性が登録されることもある。また、脆弱性の悪用という観点で見れば、悪用が容易で事業継続に多大な影響をもたらす脆弱性もあれば、条件が厳しく現実的に起こり得ないような脆弱性が登録されることもある。

そのため、システム管理者は、脆弱性が日々公表されていることを理解した上で、組織内のシステムに影響する脆弱性を適切に収集していく必要がある。

² 日本国内に加えて海外の情報も日本語に翻訳して公開をしている脆弱性対策情報のデータベース。「脆弱性対策情報データベース JVN iPedia」<https://jvndb.jvn.jp/index.html>

³ 脆弱性対策情報データベース JVN iPedia の登録状況 [2018 年第 4 四半期（10 月～12 月）]
<https://www.ipa.go.jp/security/vuln/report/JVNiPedia2018q4.html>

1.2. 昨今の脆弱性を悪用した攻撃・被害事例

脆弱性が公表されると、その情報を元に攻撃コードが作成され、攻撃が活性化する可能性がある。昨今その攻撃が行われるまでの時間が短くなっている傾向にある。本項では、脆弱性が公開された後、攻撃が活性化した事例を解説する。脆弱性対策が遅れた場合にどういった被害に遭うのかの参考にして欲しい。

1.2.1. Apache Struts2 の脆弱性を悪用した攻撃

ウェブアプリケーション開発をするためのフレームワークとして広く利用されている Apache Struts2 の脆弱性を悪用され、深刻な被害となる事例が多発している。

特に、2017年3月7日に公表された脆弱性「CVE-2017-5638」は、日本国内における複数の組織が脆弱性の悪用による不正アクセスを受け、数十万件単位のクレジットカード情報やメールアドレス等の個人情報が窃取される被害が発生した。⁴また、情報の窃取だけでなく、DoS 攻撃用のボット等のマルウェアをダウンロードおよび実行させようとする攻撃も確認されている。

2018年においても「CVE-2017-5638」に類似した脆弱性「CVE-2018-11776」を悪用する攻撃が確認された。IBM 社によると、開発ベンダが脆弱性の公表をしてから数日中に複数の実証コードが公開されたことで、攻撃者がそれらの実証コードを悪用し、仮想通貨のマイニングを目的とした攻撃を行ったと見られる兆候が確認されたと公表している。⁵

1.2.2. Oracle WebLogic Server の脆弱性を悪用した攻撃

Oracle WebLogic Server は、ウェブサイトや企業アプリケーションの構築等で広く利用されているソフトウェア製品である。

この Oracle WebLogic Server のサブコンポーネントである WLS Security に対して、既知の脆弱性「CVE-2017-10271」を悪用する攻撃が2017年12月に確認された。⁶本脆弱性に対する修正プログラムは、2017年10月時点でリリースされていたが、2017年12月下旬に実証コードが公開されたことを切っ掛けに、修正プログラムを適用していなかったシステムに対して、仮想通貨をマイニングするプログラムを仕込もうとする攻撃が確認された。

2018年4月7日には、リモートで任意のコードを実行されたり、情報を窃取されたりする可能性がある脆弱性「CVE-2018-2628」が公表された。この脆弱性も公表から2日後の4月19日に実証コードが公開されると、その脆弱性を持つシステムを探索する目的と思われる通信が急

⁴ 「Apache Struts 2」脆弱性を狙う攻撃 - 情報収集からボット感染狙いまで
<http://www.security-next.com/079352>

⁵ Apache Struts2 の脆弱性(S2-057/CVE-2018-11776)を狙う攻撃の動向について
<https://www.ibm.com/blogs/tokyo-soc/s2057/>

⁶ Oracle WebLogic Server の脆弱性 (CVE-2017-10271) を悪用する攻撃事例について
https://www.ipa.go.jp/security/ciadr/vul/20180115_WebLogicServer.html

増した。他にも、マルウェアへ感染させることを目的としたスクリプトをダウンロードさせることを目的とした攻撃の兆候も観測された。

また、同年7月17日にも、リモートで任意のコードを実行される可能性のある脆弱性「CVE-2018-2893」が公表された。2017年から活発な活動が観測されている攻撃キャンペーン「luoxk」において、脆弱性公表から4日後の7月21日に本脆弱性を悪用した攻撃が行われていることが確認されている。本攻撃事例も不正に仮想通貨のマイニングマルウェアを仕込むことが目的であった。

1.2.3. Drupal の脆弱性を悪用した攻撃

2018年4月、オープンソースのCMSであるDrupalの脆弱性「CVE-2018-7600（別名：Drupalgeddon 2.0）」を狙ったと見られる攻撃が、日本国内において数万件単位で観測された。本脆弱性が公表された直後は、目立った攻撃は確認されていなかったが、後に実証コードが公開されると、数日間で数万件単位の攻撃が確認されるようになり、ピーク時には1日で約30,000件にまで増加した。⁷ この脆弱性を悪用する攻撃によって、仮想通貨のマイニングを目的としたマルウェアに感染させられたり、システムにバックドアを設置される等の被害が発生している。

⁷ 「Drupal」脆弱性、国内で1日あたり数万件規模のアクセス - 70カ国以上から
<http://www.security-next.com/092540>

2. 脆弱性対策の考え方

本章では、自組織で管理しているサーバが脆弱性の悪用による被害に遭わないために、どのように脆弱性対策を進めればよいのかについて解説する。

なお、サーバの運用形態は、外部のデータセンターを利用するケースやクラウド上で実現するケース等、組織によって異なるため、ここではシステム管理者が自組織内でサーバを構築・管理していることを想定する。

2.1. 脆弱性対策のフロー

図 2-1-1 は、本書で解説する脆弱性対策の進め方の一例をまとめたフローである。組織によっては、より最適化された脆弱性対策の進め方があると考えられるが、本書ではこのフローを基に解説していく。

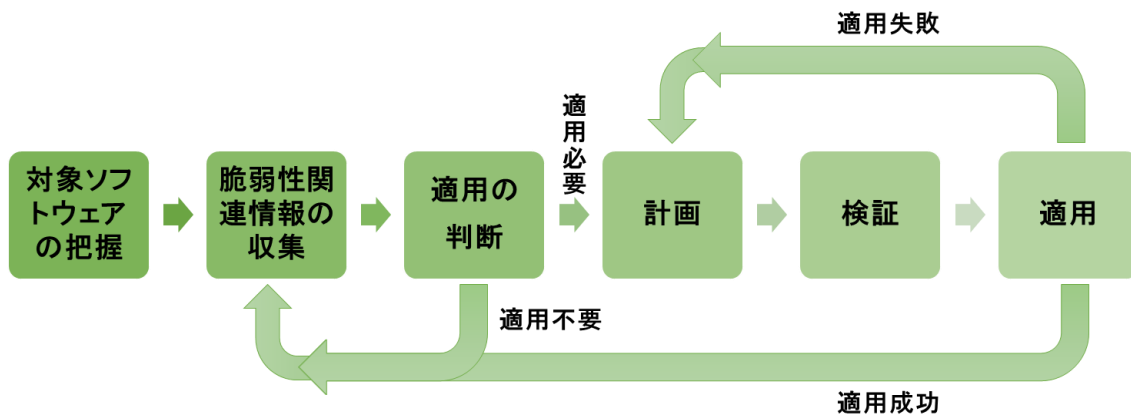


図 2-1-1 脆弱性対策のフロー

2.1.1. 対象ソフトウェアの把握

サーバ内には様々なソフトウェアがインストールされており、脆弱性対策を適切に行うためには、サーバ内にインストールされているソフトウェアを正確に把握し、管理する必要がある。そのためには最低限以下の項目を把握し、一覧で管理しておくが良い。

表 2-1-1 対象ソフトウェアの管理項目

#	項目	備考
1	ソフトウェアの名称	-
2	ソフトウェアのバージョン	-
3	ソフトウェアのインストール方法 =最新バージョンのインストール方法	【Linux サーバの場合】 yum、rpm、ソースコードからコンパイル等 【Windows サーバの場合】 インストーラ、実行ファイルの配置等
4	最新バージョンの提供サイト(URL)	最新バージョンの確認に利用するため #3 のソフトウェアのインストール方法で代替できる場合は確認不要

また、上記の項目を把握するために、サーバの構築方法に応じて以下のように対応する。

(1) 自組織のメンバーでサーバを構築した場合

- ・サーバ構築に関わったメンバーに確認する
- ・サーバ構築に関わったメンバーから提供された設計書を確認する
- ・ある程度サーバに詳しい自組織のメンバーでサーバ内を確認する

(2) 開発ベンダに委託してサーバを構築した場合

- ・開発ベンダに確認する (※1)
- ・納品された設計書を確認する

(※1) 開発ベンダに委託してサーバを構築していた場合、様々な理由により確認ができないことがある。例えば、サーバの構築に関わった開発メンバーが開発ベンダから退職してしまっているケースやサポート契約を結んでおらず確認できないケースである。そうした場合は、開発ベンダに別途有償で依頼し、仕様に関する詳細な情報を残すことを検討して欲しい。また、こうした事態に陥らないためにも、予めシステム開発時に開発ベンダへ設計書等にソフトウェアの一覧を記載するよう発注要件に含めておくことが望ましい。

2.1.2. 脆弱性関連情報の収集

脆弱性が確認されていない最新バージョンのソフトウェアや脆弱性に関する情報等を収集する。例として脆弱性関連情報の収集先には以下が考えられる。

表 2-1-2 脆弱性関連情報の収集先例

#	収集先	サイト例
1	最新バージョンの提供サイト	・ソフトウェアのダウンロードページ ・yum、rpm コマンドを利用した最新バージョンのチェック
2	脆弱性関連情報データベース	・脆弱性対策情報データベース JVN iPedia
3	ニュースサイト	・CNET ニュース：セキュリティ
4	注意喚起サイト	・IPA：重要なセキュリティ情報一覧（※2）
5	製品ベンダサイト（セキュリティアドバイザリ等）	・マイクロソフト TechNet

（※2）IPA では、組織内のウェブページに HTML タグを埋込むだけで、IPA が発信する「重要なセキュリティ情報」をリアルタイムに確認できるサイバーセキュリティ注意喚起サービス「icat（アイキャット）」を供している。是非、ご活用いただきたい。

上記の収集先のサイトを定期的を確認して、利用しているソフトウェアに関する脆弱性対策情報を収集する。なお、確認する際には少なくとも以下の観点で確認すると良い。

■確認すべき観点

- ・観点1：利用しているソフトウェアの新しいバージョンが公開されているか？
- ・観点2：利用しているソフトウェアの新しいバージョンが公開されていた場合、脆弱性対策が含まれているか？
- ・観点3：現在利用しているバージョンのソフトウェアに関する脆弱性は公開されているか？
- ・観点4：脆弱性の影響を受けるバージョンは何か？
- ・観点5：脆弱性の対策方法が公開されているか？
- ・観点6：脆弱性の深刻度（CVSS 基本値）はいくつか？
- ・観点7：ネットワーク越しの攻撃は可能か？（CVSS 基本値の評価項目から確認可）
- ・観点8：脆弱性を悪用した攻撃は行われているか？
- ・観点9：実証コードは公開されているか？

定期的な確認を行う際の周期は極力短い（毎日、2日に1回等）ほうが良い。昨今、脆弱性の公表から攻撃が開始されるまでの期間が短くなっており、確認の周期が長いと情報を確認した際には既に攻撃が始まっている恐れがある。

また、情報を収集する際には、手動で収集先のサイトを確認しても良いが、サイトの差分（追加、変更箇所）が取れるようなツールがある場合は、それらのツールを活用することで作業工数の低減を期待できる。

なお、脆弱性に関する用語(CVSS 基本値等)や上記で示したサイト以外の情報については、以下の資料の 2 章にまとめているので確認して欲しい。

参考：脆弱性対策の効果的な進め方（実践編）～ 脆弱性情報の早期把握、収集、活用のスゝメ ～

<https://www.ipa.go.jp/security/technicalwatch/20150331.html>

2.1.3. 適用の判断

収集した脆弱性関連情報の中に新しいバージョンのソフトウェア情報や脆弱性の情報、攻撃情報等を確認したら、新しいバージョンのリリースノートや CVSS 情報、緊急度を確認し、脆弱性への対応の要否を判断する。その脆弱性への対応については以下のステップで判断していく。

【ステップ1】脆弱性の存在確認

＜判断の根拠：【脆弱性関連情報の収集】の観点1、観点2、観点3、観点4、観点5＞

利用しているソフトウェアに脆弱性が存在するか否かを確認する。なお、既に脆弱性が存在することが判明している場合は、ステップ2を実施する。

なお、利用しているソフトウェアが開発元のサポート対象外のバージョンである場合、脆弱性の存在するバージョンかどうかの情報が開発ベンダから公表されない可能性があるため、可能な限り、サポート対象外のバージョンを利用しないことを推奨する。

【ステップ2】脆弱性対策の緊急度の判断

＜判断の根拠：【脆弱性関連情報の収集】の観点6、観点7、観点8、観点9＞

脆弱性に対する対策の緊急度を判断する。緊急度が高い場合は、優先度を上げて対策を実施する等の次項の「計画」段階での判断材料とする。緊急度の判断方法を以下に紹介する。

脆弱性の緊急度の判断材料として、脆弱性自身が持つ深刻度（CVSS 基本値）の高低、ネットワーク越しの攻撃の可否がある。深刻度が高い場合は、被害を受けたときの影響が大きいため緊急度は高い。さらに、ネットワーク越しに攻撃が可能である場合、世界中どこからでも攻撃される可能性があり、より緊急度が高い。続いて、脆弱性の緊急度の判断材料として、脆弱性の検証等に用いられる実証コードの存在有無や攻撃による被害の有無がある。実証コードが存在している場合、攻撃コードを容易に作成できる可能性が高いため、攻撃がすぐに発生する恐れがあり、緊急度が高い。さらに、既に攻撃が確認されている場合は、自組織もいつ被害に遭ってもおかしくない状況となるため、より緊急度が高くなる。

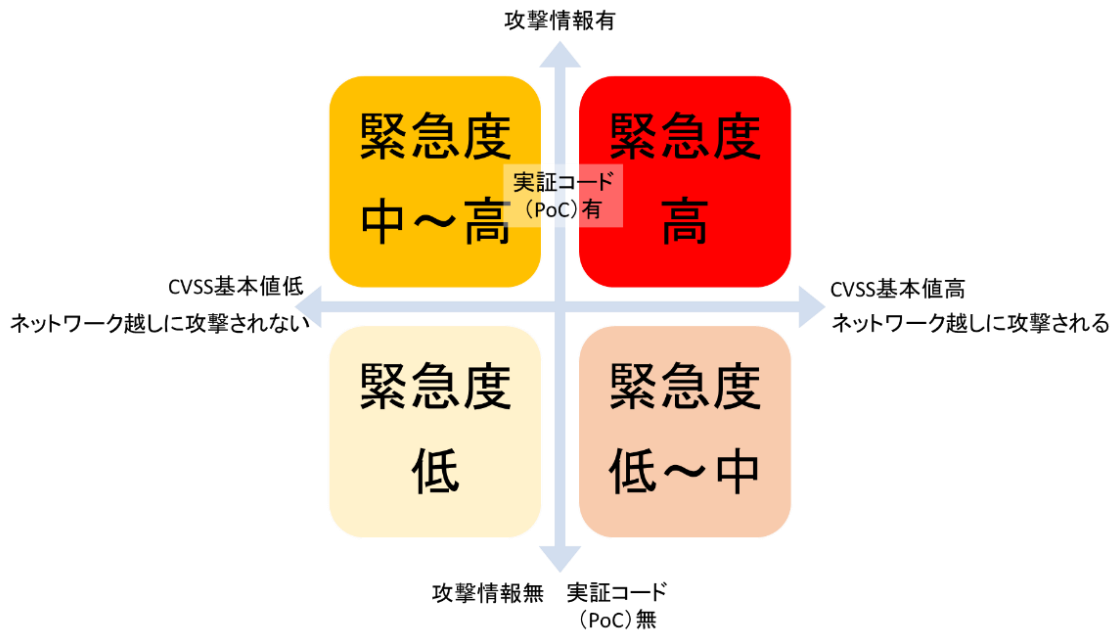


図 2-1-2 緊急度判断例

2.1.4. 計画

対策が必要な脆弱性に対して、どのように対応していくかを計画（スケジューリング）していく。なお、脆弱性の緊急度が高くても、脆弱性が存在するサーバが組織内部からしかアクセスできないような環境である等、攻撃されるリスクが低いと判断できる場合もある。また、システム環境によってはすぐに最新バージョンにアップデートできない場合もある。そのため、様々な状況を加味して計画等を立てていく。計画を立てる上でのポイント例を以下に記載する。

ポイント1：緊急度の高低

緊急度が高い場合は、可能な限り早期に対応することが望ましい。

ポイント2：サーバの環境による悪用の可能性

外部に公開しているサーバであれば、いつ攻撃されてもおかしくないため早期の対応が望ましい。逆に、組織内部からしかアクセスできないサーバであればすぐに攻撃される可能性は低い。そのため、優先度は落とすという選択肢もある。なお、優先度が低くても最終的には脆弱性の対策を行うことが望ましいため、単に対応しないとせず、月末にまとめて対応したり、数カ月から1年に掛けて徐々に対応したり等といった中長期的な観点での計画を立てる必要がある。

ポイント3：ソフトウェアのアップデートに伴う検証期間の確保

ソフトウェアをアップデートする場合、本番環境へいきなり適用すると予期せぬトラブルが発生することがある。そのため、本番環境に適用する前に検証環境でアップデートを実施し、適用の可否を判断する必要がある。その検証に、数日～数週間かかる可能性があるため、その期間を計画に含めておく必要がある。

ポイント4：脆弱性への対応方針の決定

脆弱性が存在していても、脆弱性の対策方法が存在しない場合や暫定対策しか存在しない場合がある。また、アップデートの検証の結果、事前に機能改修しなければならない等、単純にアップデートできない場合もある。対策が存在しない場合は一時的にサーバを停止させ、対策方法の公開を待つ等の対応方針を決定する。

ポイント5：ステークホルダとの合意または周知

ソフトウェアをアップデートする際、一時的にサーバの停止やサーバ上で稼働しているサービスの停止等が必要となる場合がある。その場合は、影響を与える可能性のあるステークホルダ（関係者）に同意を得る、または周知しておく必要がある。ここで適切な対応が取れていなかった場合、サーバの停止時にクレーム等のトラブルが発生する恐れがあるため、計画内に本作業を含めておく必要がある。

なお、例えば、以下のようなステークホルダがいる。

- ・サービスの組織外の利用者（一般消費者、ビジネスユーザ）
- ・サービスの組織内の利用者
- ・組織内の作業の関係者（作業承認者、サーバ室管理者等）
- ・組織内外のサーバの稼働監視作業（監視を実施している場合）
- ・ハードウェアやサーバの開発ベンダ（外部の開発ベンダに委託してサーバを構築している場合）

また、ステークホルダにより、合意を取るのか、もしくは周知のみでよいのか等の対応方法が異なるため、そこも計画しておく。

ポイント6：事業計画への影響

緊急度が高い脆弱性の場合、可能な限り早期に対応することが望ましいが、繁忙期やどうしても止められない時期等により、事業計画上リスクを許容してでも脆弱性への対応を遅らせなければならない場合もある。必要に応じてこれらを加味して計画を立てる。

2.1.5. 検証

ソフトウェアアップデート等の脆弱性対策をする場合、本番環境へいきなり適用すると予期せぬトラブルが発生することがある。そのため、本番環境に適用する前に検証環境でアップデートを実施し、適用の可否を判断する。また、検証の際に併せて脆弱性の対策手順を作成しておく。さらにその手順の中に想定しないトラブルが発生することを考慮し、元のバージョンに戻す等の切り戻し手順を含めておく。

検証環境について自組織でサーバを構築した場合は、サーバの構築に関わったメンバーに構築・協力を依頼する。外部の開発ベンダに委託してサーバを構築している場合は、発注要件に含めてサーバ構築時に併せて検証環境の作成を依頼しておく。既に開発が完了し、発注要件に含めることができない場合は、開発ベンダに有償で依頼し、構築を依頼することも検討して欲しい。

2.1.6. 適用

「2.1.4.計画」に従い、ソフトウェアアップデート等の脆弱性対策を実施する。

トラブルなく脆弱性対策が実施できた場合は、ステークホルダに対応が完了した旨を周知する。もし想定外のトラブルが発生し、脆弱性対策に失敗した場合は、【計画】に戻り、再度対応を検討する。

2.2. 自動化の重要性

これまで脆弱性対策の考え方について述べてきたが、システム管理者がサーバ内にある全てのソフトウェアを網羅的に把握し、日々公表される脆弱性の内容 1 つ 1 つを目視で確認するのは現実的ではなく、また、作業工数のコストが膨大となる恐れがある。

そのため、**自動化が可能な作業については、極力、自動化することを推奨する**。例えば、脆弱性情報の収集を自動化した場合、人的ミスによる情報の見落としや放置等を減らし、作業コストの低減につなげることができる。なお、脆弱性対策における全ての作業が自動化できるとは限らないため、自動化できる作業、できない作業を見極め、組織に適した対応を行う必要がある。また、自動化できる作業であっても、すぐの自動化が難しい場合は、最初は手動で作業を行い、順次環境を整備しつつ自動化を進めていっても良いだろう。

次章からは自動化を推進するツールの一案として、【脆弱性関連情報の収集】の一部を補完するオープンソースの脆弱性検査ツール、「Vuls (Vulnerability Scanner)」を紹介する。本ツールを通じて、組織内における作業の自動化を推進して欲しい。

3. 脆弱性検知ツール「Vuls」の紹介

3.1. 脆弱性検知ツール「Vuls」とは

Vuls (Vulnerability Scanner) は、Linux/FreeBSD 系サーバシステム向けに開発された脆弱性関連情報の収集と検知を自動化する脆弱性検知ツールで、2016年4月からオープンソースソフトウェアとして Github⁸上に公開されている。

このツールは、組織内で管理しているサーバにインストールされているソフトウェアの情報と、脆弱性データベース (NVD⁹や JVN iPedia) に登録されている脆弱性対策情報を関連付けることで、インストールされているソフトウェアに影響する脆弱性対策情報のみを検知し、一元管理できる (図 3-1-1)。また、検知した脆弱性はターミナルツール上やブラウザ上で確認したり、脆弱性の深刻度に応じて、メールや Slack¹⁰等を使い利用者に通知することが可能である。本ツールを活用することで、Vuls の利用者 (システム管理者を想定) は、日々の脆弱性情報の収集に掛かる工数を低減させ、第 2 章で述べた脆弱性対策において行うべき作業の円滑化が期待できる。

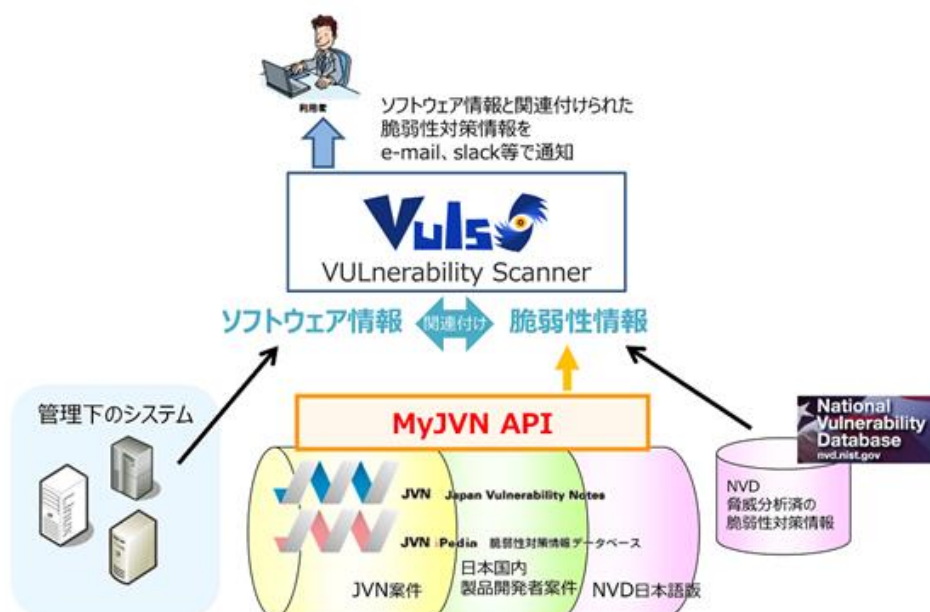


図 3-1-1 脆弱性検知ツール「Vuls」の利用イメージ

⁸ Github とは、ソフトウェア開発のプラットフォームのこと。 <https://github.co.jp/>

⁹ NVD とは、National Vulnerability Database : NIST が運営する脆弱性データベースのこと。 <https://nvd.nist.gov/>

¹⁰ Slack とは、ビジネスチャットアプリのこと。 <https://slack.com/>

3.2. 特徴

Vuls の特徴を説明する。

(1) 脆弱性関連情報の収集・検知を自動化

- ・利用者が管理するシステムに影響を与える脆弱性のみを自動的に検出するため、脆弱性関連情報の収集に掛かる工数を低減できる。
- ・OS の機能である cron を使い、定期的に脆弱性検知を行うことで、脆弱性情報の収集漏れを防げる。
- ・脆弱性情報に記載されている深刻度レベルが閾値を超えたものを、メールや Slack に自動送付することで、脆弱性情報を確認する工数を低減できる。

(2) 日本語への対応

- ・脆弱性関連情報の収集元として「脆弱性対策情報データベース JVN iPedia」に対応しており、検知結果を日本語で確認できる。

(3) 脆弱性検知対象のシステムにエージェントは不要

- ・エージェントレスアーキテクチャを採用しているため、Vuls を導入したサーバに一連のソフトウェアをインストールするだけで、複数のシステムの脆弱性を検知できる。

(4) Linux/FreeBSD 系の複数の OS をサポート

- ・Vuls は、表 3-2-1 にある Linux/FreeBSD 系の OS をサポート¹¹しており、これらの環境上で脆弱性検知を実行可能である。また、オンプレミス環境だけではなく、クラウド環境での実行にも対応している。

表 3-2-1 サポート対象の OS とバージョン (2019 年 1 月時点)

サポート OS	バージョン
Alpine	3.3 以降
Ubuntu	14, 16, 18
Debian	8, 9
RHEL	5, 6, 7
Oracle Linux	5, 6, 7
CentOS	6, 7
Amazon Linux	全て
FreeBSD	10, 11
SUSE Enterprise	11, 12
Aspbian	Jessie, Stretch

¹¹ Supported OS

<https://vuls.io/docs/en/supported-os.html>

(5) 様々なインストール形態への対応

- ・Vuls が検知可能なソフトウェアは、OS のパッケージマネージャで取得したソフトウェア (.rpm や .deb 等) を対象としているが、脆弱性検知したいソフトウェアが CPE (Common Platform Enumeration)¹² に登録されていれば、プログラミング言語のフレームワークやライブラリ、ミドルウェア等も検知できる。
- 詳細については以下を参照すること。

➤ Scan vulnerabilities of non-OS packages

<https://vuls.io/docs/ja/usage-scan-non-os-packages.html>

(6) 他ソフトウェアとの連携が豊富

- ・サーバの監視ツールである「Zabbix¹³」や「Deep Security¹⁴」にも対応しており、利用者はこれらのツール上で Vuls の脆弱性検知結果を確認できる。

・ Docker コンテナ¹⁵に対応しており、Docker を利用できる環境であれば、よりスムーズにインストール作業を進めることが出来る。

→インストール方法については「4.6. (参考) Docker インストール」を参照すること。

- ・VulsRepo や AWS Quicksight¹⁶等といった様々な WebUI で脆弱性情報を確認できる。

➤ @VulsRepo

<https://vuls.io/docs/en/vulsrepo.html>

<http://usi360.github.io/vulsrepo/#drawerLeft>

VulsRepo の利用イメージを GIF 形式または Web ページ上で確認可能。

¹² 共通プラットフォーム一覧 CPE 概説

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

¹³ Zabbix とは、ネットワーク管理ソフトウェアのこと。 <https://www.zabbix.com/jp/>

¹⁴ Deep Security とは、様々な機能を持つ統合型セキュリティ製品のこと。

https://www.trendmicro.com/ja_jp/business/products/hybrid-cloud/deep-security-data-center.html

¹⁵ Docker とは、コンテナ型の仮想化環境を提供するオープンソースソフトウェアのこと。

<https://www.docker.com/>

¹⁶ AWS Quicksight とは、クラウド駆動の高速なビジネスインテリジェンスサービスののこと。

<https://aws.amazon.com/jp/quicksight/>

4. Vulns 動作検証

ここでは筆者が実際に Vulns をインストールし、動作検証を行った結果について記載していく。なお、本書に記載している各ソフトウェアのバージョンや URL 等は、2018 年 11 月時点で確認したものであるため、インストールする際は最新のバージョンを利用することを推奨する。

4.1. 脆弱性検知ツール実行環境およびバージョン

(1) 使用した各種ソフトウェアの名称及びバージョンについて以下に示す。

表 4-1-1 使用した OS、ソフトウェアとバージョン情報

OS 名	バージョン
CentOS (Vulns 管理、スキャン対象サーバ)	7.5.1804
ソフトウェア名	バージョン
SQLite3 (sqlite)	3.7.17-8.el7
Git	1.8.3.1-14.el7_5
Gcc	4.8.5-28.el7_5.1
GNU Make (make)	1:3.82-23.el7
Wget	1.14-15.el7_4.1
yum-utils	1.1.31-46.el7_5
Go	1.11.2
go-cve-dictionary	v0.3.1 b083bed
goval-dictionary	v0.1.0 0072c14
gost	920046a
go-exploitdb	-
Vulns	v0.6.1 8eae500
VulnsRepo	v0.2.0

(2) 使用したソフトウェアのインストール先パスについて以下に示す。

表 4-1-2 脆弱性検知ツール Vuls 等のディレクトリ

ソフトウェア名	パス情報
vuls	\$GOPATH/src/github.com/future-architect/vuls
go-cve-dictionary	\$GOPATH/src/github.com/kotakanbe/go-cve-dictionary
goval-dictionary	\$GOPATH/src/github.com/kotakanbe/goval-dictionary
gost	\$GOPATH/src/github.com/knqyf263/gost
go-exploitdb	\$GOPATH/src/github.com/mozqnet/go-exploitdb
VulsRepo	\$HOME/vulsrepo

※\$GOPATH=\$HOME/go、\$HOME=/home/vulsuser

(3) Vuls を実行する際に使用したユーザと権限について以下に示す。

表 4-1-3 脆弱性検知ツール Vuls 実行に使ったユーザ名および権限

サーバ名/ (使用 OS)	ユーザ名	権限
Vuls 管理サーバ (CentOS 7)	vulsuser	sudo
スキャン対象サーバ (CentOS 7)	scanuser	sudo

(4) 本書で Vuls を検証した環境について以下に示す。

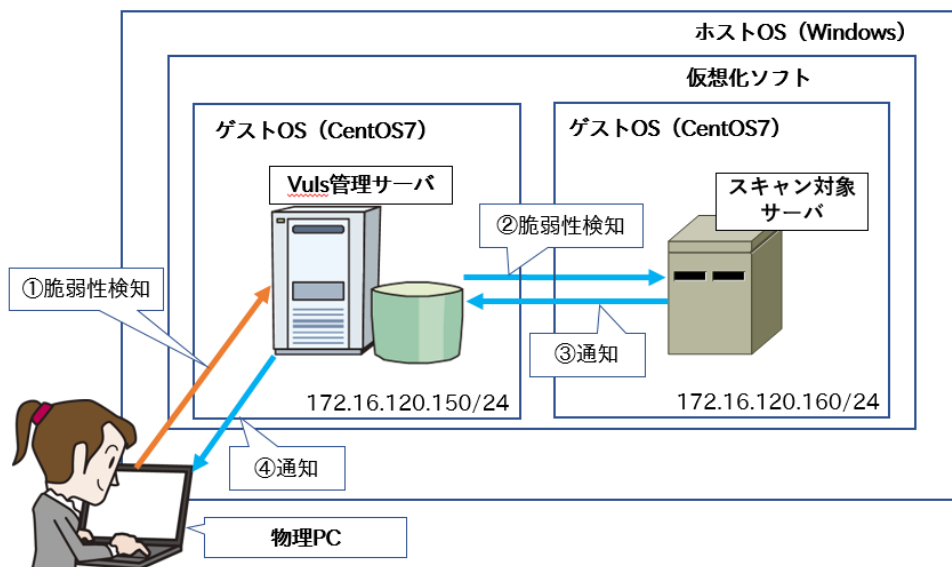


図 4-1-1 検証環境の構成図

4.2. 注意事項

(1) 詳細を確認したい場合は開発者のウェブサイト等を参照する

本書に記載している手順は、筆者が Vuls に必要となるソフトウェアをインストールし、実際に脆弱性検知の動作検証をするまでを示したものである。そのため、各インストールソフトウェアの用途や機能についての解説はしない。必要に応じて開発者のウェブサイトを参照して欲しい。

(2) インストール方法は「4.5.手動インストール」のみ検証

2018 年 11 月時点における Vuls のインストール方法は、以下の 3 つが案内されている。

表 4-2-1 各インストール方法の概要

インストール方法	概要
手動インストール	Linux パッケージ (Git や make 等) を利用してインストールする方法
Docker インストール	アプリケーションとその実行環境、スクリプト等をパッケージ化した「Docker イメージ」をインストールする方法
awless インストール	AWS を管理するためのコマンドラインインターフェース (CUI) である「awless」を利用してインストールする方法

本書では上記 3 つの内、手動インストールによる検証を実施した。なお、読者が Docker を利用できる環境を持つ場合、必要となるソフトウェアを比較的容易にインストールすることができるので、「4.6. (参考) Docker インストール」を参照して欲しい。

4.3. 動作検証の流れ

Vuls の検証は以下の流れで行う。

- ① 事前準備
OS の設定や Vuls をインストールする前に必要なソフトウェアのインストールを行う。
- ② Vuls のインストールおよび初期設定
Vuls および関連ソフトウェアのインストールを行う。なお、参考として、Docker を使ったインストール方法も簡易的に試している。
- ③ 脆弱性の検知（手動、自動）
Vuls を実行し、脆弱性検知を行う。またスケジューラ（cron）を利用して定期的に脆弱性検知を行う。
- ④ レポートの確認
脆弱性検知結果レポートを確認する。また、WebUI の VulsRepo によるレポートを確認する。

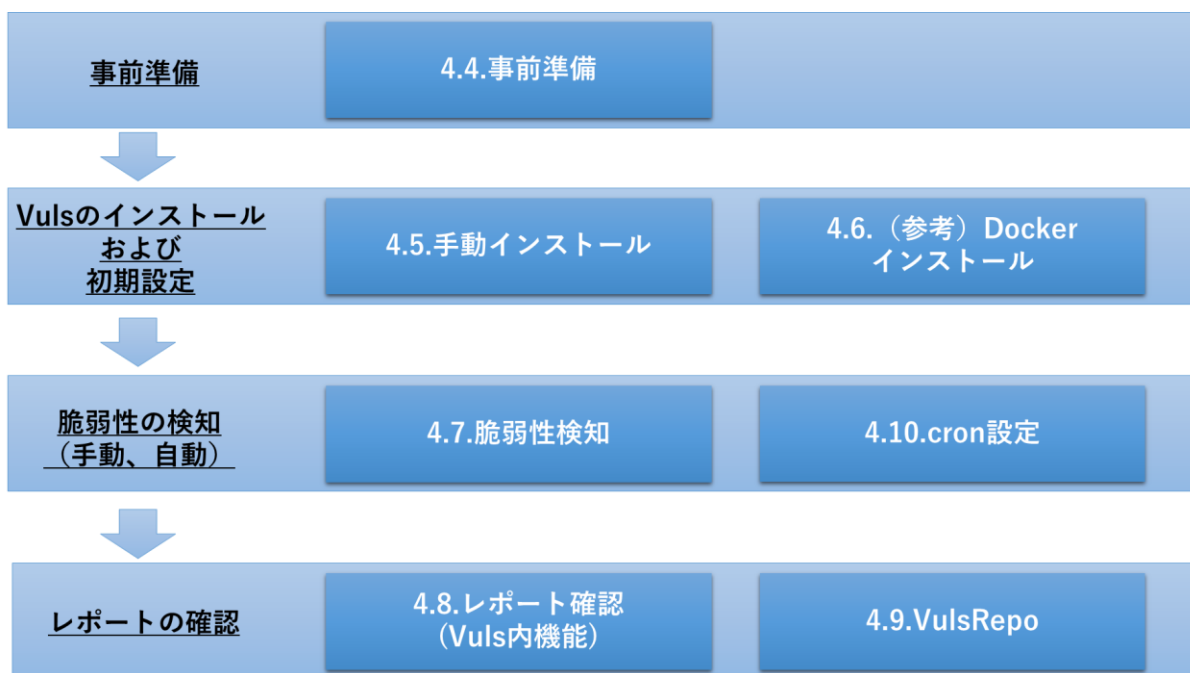


図 4-3-1 脆弱性検知ツール Vuls の検証の流れ

4.4. 事前準備

Vuls をインストールする前の準備として、表 4-4-1 にあるソフトウェアをインストールする。

表 4-4-1 事前に準備するソフトウェアの一覧

ソフトウェア名	ベンダページ
SQLite3 (sqlite)	https://www.sqlite.org/index.html
Git	https://git-scm.com/
GCC	https://gcc.gnu.org/
GNU Make (make)	https://www.gnu.org/software/make/
Wget	https://www.gnu.org/software/wget/
yum-utils ※スキャン対象サーバにも必要	https://centos.pkgs.org/7/centos-x86_64/yum-utils-1.1.31-45.el7.noarch.rpm.html
Go ※最新バージョンを推奨	https://golang.org/doc/install

- ① Vuls 管理サーバにログイン（または、SSH 接続）する。
- ② root 権限に昇格する。
※外部ネットワークへ接続が必要となるため、適宜プロキシの設定等を行っておくこと。

●参考：proxy 認証設定

(1) /etc 配下の設定ファイルを開き、最終行に以下を追加する。

```
# vi /etc/profile
```

```
PROXY='(プロキシサーバの IP アドレス):(ポート番号)'  
export http_proxy=$PROXY  
export HTTP_PROXY=$PROXY  
export https_proxy=$PROXY  
export HTTPS_PROXY=$PROXY  
PROXY_USERNAME=(プロキシ認証されたユーザ名)  
PROXY_PASSWORD=(プロキシ認証されたユーザのパスワード)
```

(2) 設定を反映する。

```
# source /etc/profile
```

- ③ Vuls を実行するユーザ（本書では「vulsuser」）を作成する。

```
# useradd vulsuser
```

- ④ 「vulsuser」にパスワードを設定する。

```
# passwd vulsuser
```

⑤ 一般ユーザの「vulsuser」が root 権限のコマンドも実行できるよう設定に追記する。

```
# visudo
```

```
## Allow root to run any commands anywhere
root    ALL=(ALL)          ALL
vulsuser    ALL=(ALL)      ALL    (←新規追加)
```

⑥ su コマンドを使い「vulsuser」にユーザ変更する。

```
# su - vulsuser
```

⑦ yum を使い SQLite3・Git・GCC・GNU Make・Wget・yum-utils をインストールする。

```
$ sudo yum -y install sqlite git gcc make wget yum-utils
```

⑧ Wget を使い go1.11.2.linux-amd64.tar.gz を取得する。

```
$ wget https://dl.google.com/go/go1.11.2.linux-amd64.tar.gz
```

⑨ ⑧で取得した go1.11.2.linux-amd64.tar.gz を/home 配下に解凍する。

```
$ sudo tar -C /usr/local -xzf go1.11.2.linux-amd64.tar.gz
```

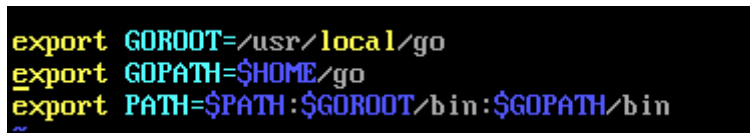
⑩ /home 配下に go ディレクトリを作成する。

```
$ mkdir $HOME/go
```

⑪ vi エディタで環境設定ファイルである goenv.sh 内に以下の 3 行を追加する。

```
$ sudo vi /etc/profile.d/goenv.sh
```

```
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```



```
export GOROOT=/usr/local/go
export GOPATH=$HOME/go
export PATH=$PATH:$GOROOT/bin:$GOPATH/bin
```

図 4-4-1 goenv.sh 設定内容

⑫ ⑪で設定した環境変数を反映する。

```
$ source /etc/profile.d/goenv.sh
```

4.5. 手動インストール

Vuls を利用するために必要なソフトウェアを手動でインストールする。

4.5.1. go-cve-dictionary をインストールする

「go-cve-dictionary」は、脆弱性データベース（NVD や JVN iPedia）に登録されている脆弱性情報を Vuls 管理サーバ内に取り込み、管理するためのツールである。

■go-cve-dictionary :

```
https://github.com/kotakanbe/go-cve-dictionary
```

① /var/log 配下に vuls ディレクトリを作成する。

```
$ sudo mkdir /var/log/vuls
```

② vuls ディレクトリの所有者を「vulsuser」に変更する。

```
$ sudo chown vulsuser /var/log/vuls
```

③ vulsuser に vuls ディレクトリのフルコントロールを与える。

```
$ sudo chmod 700 /var/log/vuls
```

④ \$GOPATH/src/github.com 配下に kotakanbe ディレクトリを作成する。

```
$ mkdir -p $GOPATH/src/github.com/kotakanbe
```

⑤ \$GOPATH/src/github.com/kotakanbe に移動する。

```
$ cd $GOPATH/src/github.com/kotakanbe
```

⑥ git を使い go-cve-dictionary.git を取得する。

```
$ git clone https://github.com/kotakanbe/go-cve-dictionary.git
```

⑦ /go-cve-dictionary に移動する。

```
$ cd go-cve-dictionary
```

⑧ go-cve-dictionary をインストールする。

```
$ make install
```

⑨ \$GOPATH/bin 配下に go-cve-dictionary のバイナリファイルが生成されることを確認する。

```
$ cd $GOPATH/bin
```



```
$ ls go-cve-dictionary
```

```
[vulsuser@localhost bin]$ ls go-cve-dictionary  
go-cve-dictionary
```

図 4-5-1 go-cve-dictionary のバイナリファイル確認

●参考：go-cve-dictionary のバージョン確認方法

```
$ cd $HOME
```

```
$ go-cve-dictionary -v
```

```
[vulsuser@localhost ~]$ go-cve-dictionary -v  
go-cve-dictionary v0.3.1 b083bed
```

図 4-5-2 go-cve-dictionary のバージョン確認

4.5.2. 脆弱性データベースから脆弱性情報を取得する

本書では日本語（JVN iPedia）と英語（NVD）の脆弱性情報を確認するために、①と②を両方とも実行している。

① JVN iPedia より 1998 年から作業年までの脆弱性情報を取得する。

（※作業時間は利用者の環境によるが、およそ 10 分程度掛かる。）

```
$ cd $HOME
```

```
$ for i in `seq 1998 $(date +%Y)`; do go-cve-dictionary fetchjvn -years $i; done
```

② NVD より 2002 年から作業年までの脆弱性情報を取得する。

（※作業時間は利用者の環境によるが、およそ 15 分程度掛かる。）

```
$ cd $HOME
```

```
$ for i in `seq 2002 $(date +%Y)`; do go-cve-dictionary fetchnvd -years $i; done
```

③ ①と②の情報が正しく取得できたか確認する。検証では合計で 1.9G 取得している。

```
$ ls -alh cve.sqlite3
```

```
[vulsuser@localhost ~]$ ls -alh cve.sqlite3  
-rw-rw-r--. 1 vulsuser vulsuser 1.9G 11月 28 19:56 cve.sqlite3
```

図 4-5-3 go-cve-dictionary のデータ確認

4.5.3. goval-dictionary をインストールする

「goval-dictionary」は、OS の OVAL 情報¹⁷を Vuls 管理サーバ内に取り込み、管理するためのツールである。

■ goval-dictionary :

```
https://github.com/kotakanbe/goval-dictionary
```

① \$GOPATH/src/github.com 配下に kotakanbe ディレクトリを作成する。

```
$ mkdir -p $GOPATH/src/github.com/kotakanbe
```

② \$GOPATH/src/github.com/kotakanbe に移動する。

```
$ cd $GOPATH/src/github.com/kotakanbe
```

③ git を使い goval-dictionary.git をコピーする。

```
$ git clone https://github.com/kotakanbe/goval-dictionary.git
```

④ /goval-dictionary に移動する。

```
$ cd goval-dictionary
```

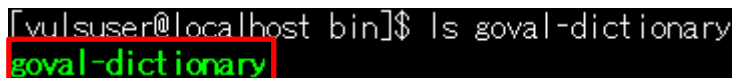
⑤ goval-dictionary をインストールする。

```
$ make install
```

⑥ \$GOPATH/bin 配下にバイナリが作成されることを確認する。

```
$ cd $GOPATH/bin
```

```
$ ls goval-dictionary
```



```
[vulsuser@localhost bin]$ ls goval-dictionary  
goval-dictionary
```

図 4-5-4 goval-dictionary のバイナリファイル確認

¹⁷ セキュリティ検査言語 OVAL 概説

<https://www.ipa.go.jp/security/vuln/OVAL.html>

●参考：goval-dictionary のバージョン確認方法

```
$ cd $HOME
```

```
$ goval-dictionary -v
```

```
[vulsuser@localhost ~]$ goval-dictionary -v  
goval-dictionary v0.1.0 0072c14
```

図 4-5-5 goval-dictionary のバージョン確認

4.5.4. Redhat から OVAL 情報を取得する

OS の OVAL 情報を取得する。

本書では CentOS 7 をインストールしたサーバに対して脆弱性検知を行うため、CentOS と共通の OVAL 情報となる Red Hat から収集する。なお、他のサポート OS に対して脆弱性検知を行う場合は、表 4-5-1 より対応するサポート OS とバージョンに合った OVAL 情報を取得する。

① Red Hat 7 の情報を goval-dictionary に蓄積する。

```
$ cd $HOME
```

```
$ goval-dictionary fetch-redhat 7
```

(※作業時間は利用者の環境によるが、およそ 3 分程度掛かる。)

表 4-5-1 OVAL 情報の収集先一覧

OS	OVAL 情報の収集先
Alpine	https://github.com/kotakanbe/goval-dictionary#usage-fetch-alpine-secdb-as-oval-data-type
Red Hat CentOS	https://github.com/kotakanbe/goval-dictionary#usage-fetch-oval-data-from-redhat
Debian	https://github.com/kotakanbe/goval-dictionary#usage-fetch-oval-data-from-debian
Ubuntu	https://github.com/kotakanbe/goval-dictionary#usage-fetch-oval-data-from-ubuntu
Oracle Linux	https://github.com/kotakanbe/goval-dictionary#usage-fetch-oval-data-from-oracle
SUSE	https://github.com/kotakanbe/goval-dictionary#usage-fetch-oval-data-from-suse

4.5.5. gost をインストールする

「gost」は、Redhat や Debian 等で公開された脆弱性情報を取得するためのツールである。

■gost :

```
https://github.com/knqyf263/gost
```

① /var/log 配下に gost ディレクトリを作成する。

```
$ sudo mkdir /var/log/gost
```

② gost ディレクトリの所有者を vulsuser に変更する。

```
$ sudo chown vulsuser /var/log/gost
```

③ vulsuser に gost ディレクトリのフルコントロールを与える。

```
$ sudo chmod 700 /var/log/gost
```

④ \$GOPATH/src/github.com 配下に knqyf263 ディレクトリを作成する。

```
$ mkdir -p $GOPATH/src/github.com/knqyf263
```

⑤ knqyf263 に移動する。

```
$ cd $GOPATH/src/github.com/knqyf263
```

⑥ git を使い gost.git をインストールする。

```
$ git clone https://github.com/knqyf263/gost.git
```

⑦ gost に移動する。

```
$ cd gost
```

⑧ gost をインストールする。

```
$ make install
```

⑨ \$GOPATH/bin 配下にバイナリが作成されることを確認する。

```
$ cd $GOPATH/bin
```

```
$ ls gost
```

```
[vulsuser@localhost bin]$ ls gost  
gost
```

図 4-5-6 gost のバイナリファイル確認

●参考：gost のバージョン確認方法

```
$ cd $HOME
```

```
$ gost -v
```

```
[vulsuser@localhost ~]$ gost -v  
gost 920046a
```

図 4-5-7 gost のバージョン情報

4.5.6. Redhat から脆弱性情報を取得する

Redhat が登録した脆弱性情報を取得する。

なお、筆者が実行した際、取得に時間が掛かったため、ある程度期間を限定し、2016年1月1日から作業時点までの情報を取得する。

① Red Hat の情報を gost に蓄積する。

```
$ gost fetch redhat --after 2016-01-01
```

(※作業時間は利用者の環境によるが、およそ 25 分程度掛かる。)

(※組織内に proxy サーバが存在する場合、HTTP エラーが発生する可能性がある。その場合は以下のコマンドを使用すること。)

② Red Hat の情報を proxy を介して、gost に蓄積する。

```
$ gost fetch redhat --after 2016-01-01 --http-proxy http:// (プロキシサーバの IP アドレス) :  
(ポート番号)
```

4.5.7. go-exploitdb をインストールする

脆弱性検知ツール Vuls の 0.6.0 バージョンから ExploitDB.com を情報元として、検出された CVE に関する実証コードを検出可能できるようになった。go-exploitdb はその情報を取り込み、管理するためのツールである。なお、本書では手順を記載するのみとする。

■go exploitdb :

```
https://github.com/mozqnet/go-exploitdb
```

① /var/log 配下に go-exploitdb ディレクトリを作成する。

```
$ sudo mkdir /var/log/go-exploitdb
```

② go-exploitdb ディレクトリの所有者を vulsuser に変更する。

```
$ sudo chown vulsuser /var/log/go-exploitdb
```

③ vulsuser に go-exploitdb ディレクトリのフルコントロールを与える。

```
$ sudo chmod 700 /var/log/go-exploitdb
```

④ \$GOPATH/src/github.com 配下に mozqnet ディレクトリを作成する。

```
$ mkdir -p $GOPATH/src/github.com/mozqnet
```

⑤ mozqnet に移動する。

```
$ cd $GOPATH/src/github.com/mozqnet
```

⑥ git を使い go-exploitdb.git をインストールする。

```
$ git clone https://github.com/mozqnet/go-exploitdb.git
```

⑦ go-exploitdb に移動する。

```
$ cd go-exploitdb
```

⑧ go-exploitdb をインストールする。

```
$ make install
```

⑨ \$GOPATH/bin 配下にバイナリが作成されることを確認する。

```
$ cd $GOPATH/bin
```

```
$ ls go-exploitdb
```

```
[vulsuser@localhost bin]$ ls go-exploitdb  
go-exploitdb
```

図 4-5-8 go-exploitdb のバイナリファイル確認

⑩ ExploitDB から情報を入手する。

```
$ go-exploitdb fetch
```

(※組織内に proxy サーバが存在する場合、HTTP エラーが発生する可能性がある。その場合は以下のコマンドを使用すること。)

③ ExploitDB.com の情報を proxy を介して、go-exploitdb に蓄積する。

```
$ go-exploitdb fetch --http-proxy http:// (プロキシサーバの IP アドレス) : (ポート番号)
```

4.5.8. Vuls をインストールする。

脆弱性検知ツール Vuls をインストールする。Vuls が既に再インストールされている場合は、⑦以降の手順を実施する。

■Vuls :

```
https://github.com/future-architect/vuls
```

① \$GOPATH/src/github.com 配下に future-architect ディレクトリを作成する。

```
$ mkdir -p $GOPATH/src/github.com/future-architect
```

② future-architect に移動する。

```
$ cd $GOPATH/src/github.com/future-architect
```

③ git を使い vuls.git をインストールする。

```
$ git clone https://github.com/future-architect/vuls.git
```

④ vuls に移動する

```
$ cd vuls
```

⑤ Vuls をインストールする。

```
$ make install
```

⑥ \$GOPATH/bin 配下にバイナリが作成されることを確認する。

```
$ cd $GOPATH/bin
```

```
$ ls vuls
```

```
[vulsuser@localhost bin]$ ls vuls  
vuls
```

図 4-5-9 Vuls のバイナリファイル確認

●参考：Vuls のバージョン確認方法

```
$ cd $HOME
```

```
$ vuls -v
```

```
[vulsuser@localhost ~]$ vuls -v  
vuls v0.6.1 8eae500
```

図 4-5-10 Vuls のバージョン確認

※過去に Vuls をインストールしたことがある、または Vuls を更新したい場合に関しては、以下の再インストール手順を実施する。

- ⑦ \$GOPATH/pkg/linux_amd64/github.com/future-architect 配下にある vuls ディレクトリを削除する。

```
$ rm -rf $GOPATH/pkg/linux_amd64/github.com/future-architect/vuls
```

- ⑧ \$GOPATH/src/github.com/future-architect 配下の vuls ディレクトリを削除する。

```
$ rm -rf $GOPATH/src/github.com/future-architect/vuls/
```

- ⑨ \$GOPATH/src/github.com/future-architect に移動する。

```
$ cd $GOPATH/src/github.com/future-architect
```

- ⑩ git を使い vuls.git をインストールする。

```
$ git clone https://github.com/future-architect/vuls.git
```

- ⑪ vuls に移動する。

```
$ cd Vuls
```

- ⑫ vuls を再インストールする

```
$ make install
```

- ⑬ ⑫まで実施すると、\$GOPATH/bin 配下にバイナリが作成される。

```
$ cd $GOPATH/bin
```

```
$ ls vuls
```

```
[vulsuser@localhost bin]$ ls vuls  
vuls
```

図 4-5-11 Vuls のバイナリファイル確認

4.6. (参考) Docker インストール

本項では、Docker を使ったインストール方法を簡易的に紹介する。各ソフトウェアの詳細については「4.5.手動インストール」を参照して欲しい。

表 4-6-1 Docker を使用した各ソフトウェアの詳細情報

ソフトウェア名	詳細情報
go-cve-dictionary	https://hub.docker.com/r/vuls/go-cve-dictionary/
goval-dictionary	https://hub.docker.com/r/vuls/goval-dictionary/
gost	https://hub.docker.com/r/vuls/gost/
vuls	https://hub.docker.com/r/vuls/vuls/

4.6.1. 前提条件

Docker を使う上での前提条件については、以下の開発者ページを参照すること。

➤ @Docker

<https://vuls.io/docs/ja/install-with-docker.html>

4.6.2. セットアップ

Vuls の Docker イメージはコミットする度に作成され、Docker/Vuls 内に蓄積される。

① Docker で go-cve-dictionary をインストールする。

```
$ docker pull vuls/go-cve-dictionary
```

② Docker で goval-dictionary をインストールする。

```
$ docker pull vuls/goval-dictionary
```

③ Docker で gost をインストールする。

```
$ docker pull vuls/gost
```

④ Docker で Vuls をインストールする。

```
$ docker pull vuls/vuls
```

4.6.3. ソフトウェアのバージョン情報を確認する

- ① go-cve-dictionary のバージョンを確認する。

```
$ docker run --rm vuls/go-cve-dictionary -v
```

- ② goval-dictionary のバージョンを確認する。

```
$ docker run --rm vuls/goval-dictionary -v
```

- ③ gost のバージョンを確認する。

```
$ docker run --rm vuls/gost -v
```

- ④ vuls のバージョンを確認する。

```
$ docker run --rm vuls/vuls -v
```

4.7. 脆弱性検知

Vuls で実行できる脆弱性検知には、以下の 2 種類が用意されている。

表 4-7-1 スキャンの種類 1

スキャン名	詳細
Local Scan Mode	Vuls を導入したサーバ自身に対して行う脆弱性検知
Remote Scan Mode	Vuls 管理サーバからリモート (SSH) 接続したスキャン対象サーバに対して行う脆弱性検知

また、表 4-7-1 のいずれかを実行する際、以下の 4 つの中から 1 つを選択して実行できる。

表 4-7-2 スキャンの種類 2

スキャン名	詳細
fast-scan	ルート権限を使用しない簡易的な脆弱性検知。スキャンするサーバにほとんど負荷をかけることなく実行可能
fast-root scan mode	ルート権限で行う簡易的な脆弱性検知。スキャンするサーバにほとんど負荷をかけることなく実行可能
Deep scan mode	ルート権限で行う詳細な脆弱性検知。yum changelog を発行して、アップグレード可能なパッケージの変更ログを取得・解析し、更新が必要なソフトウェアの一覧を取得することが可能
Offline scan mode	fast-scan、fast-root scan mode を実行する際に、-offline コマンドを付与することで利用可能。インターネット接続できない環境で有用（脆弱性情報等は別サーバからコピーする）

なお、本書の検証では、Vuls の最も簡易なスキャンモードである「fast-scan」を用いる。他のスキャンモードを実行する場合は、以下を参照すること。

➤ Scan

<https://vuls.io/docs/ja/usage-scan.html>

4.7.1. Local Scan Mode

Vuls を導入したサーバ自身に対してスキャンを行う「Local Scan Mode」の利用方法について検証した。なお、脆弱性検知を実施した際に出力される脆弱性レポートの見方については「4.8.レポート確認 (Vuls 内機能)」を参照のこと。

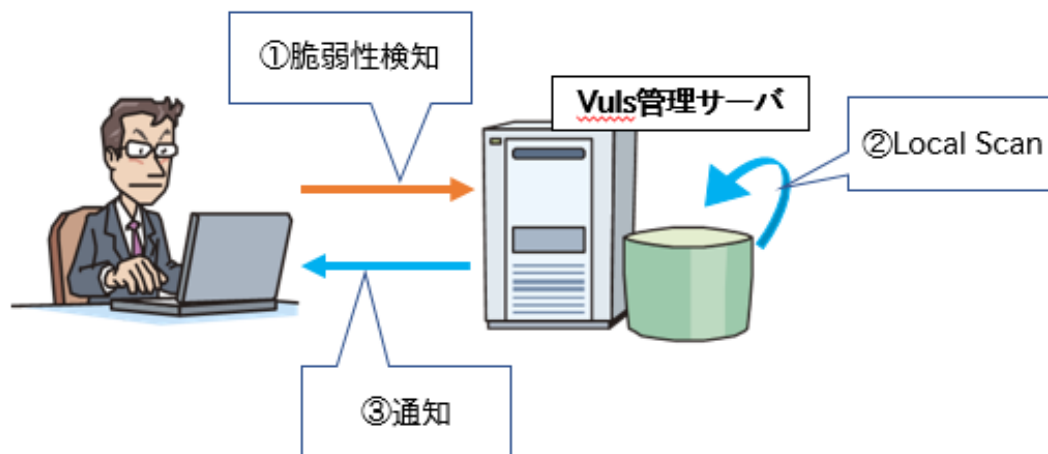


図 4-7-1 Local Scan Mode のイメージ

① 「4.5.手動インストール」または「4.6. (参考) Docker インストール」でセットアップした Vuls 管理サーバを用意する。

② \$HOME に移動する。

```
$ cd $HOME
```

③ vi エディタで「config.toml」を作成し、以下の設定を行う。

```
$ vi config.toml
```

```
[servers]
```

```
[servers.localhost]
```

```
host = "localhost"
```

```
port = "local"
```

```
[servers]

[servers.localhost]
host = "localhost"
port = "local"
```

図 4-7-2 config.toml-localhost 設定内容

④ ③で設定した config.toml ファイルと、Vuls 管理サーバの設定状況を確認する。

```
$ vuls configtest
```

```
[vulsuser@localhost ~]# vuls configtest
[Nov 18 19:15:34] INFO [localhost] Validating config...
[Nov 18 19:15:34] INFO [localhost] Detecting Server/Container OS...
[Nov 18 19:15:35] INFO [localhost] Detecting OS of servers...
[Nov 18 19:15:35] INFO [localhost] (1/1) Detected: localhost: centos 7.5.1804
[Nov 18 19:15:35] INFO [localhost] Detecting OS of containers...
[Nov 18 19:15:35] INFO [localhost] Checking Scan Modes...
[Nov 18 19:15:35] INFO [localhost] Checking dependencies...
[Nov 18 19:15:35] INFO [localhost] Dependencies ... Pass
[Nov 18 19:15:35] INFO [localhost] Checking sudo settings...
[Nov 18 19:15:35] INFO [localhost] Sudo... Pass
[Nov 18 19:15:35] INFO [localhost] It can be scanned with fast scan mode even if v
ettings in fast-root or deep scan mode
[Nov 18 19:15:35] INFO [localhost] Scannable servers are below...
localhost
[vulsuser@localhost ~]# _
```

図 4-7-3 vuls configtest 実行結果

⑤ 脆弱性検知を実行する。

```
$ vuls scan
```

```
One Line Summary
=====
localhost      centos7.5.1804  365 installed, 4 updatable
```

図 4-7-4 Local Scan Mode 実行結果例

実行結果の詳細は「4.8.レポート確認 (Vuls 内機能)」を参照すること。

4.7.2. Remote Scan Mode

Vuls 管理サーバとリモート接続 (SSH) をしたスキャン対象サーバに対して脆弱性検知する「Remote Scan Mode」の利用方法について検証した。なお、「4.7.1.Local Scan Mode」同様、脆弱性レポートの見方については、「4.8.レポート確認 (Vuls 内機能)」を参照して欲しい。

また、Remote Scan Mode を利用する上での注意点として、Vuls はリモート接続におけるパスワード認証をサポートしていないため、Vuls 管理サーバ側で鍵ペアを生成し、スキャン対象サーバに公開鍵を設定しておく必要がある。

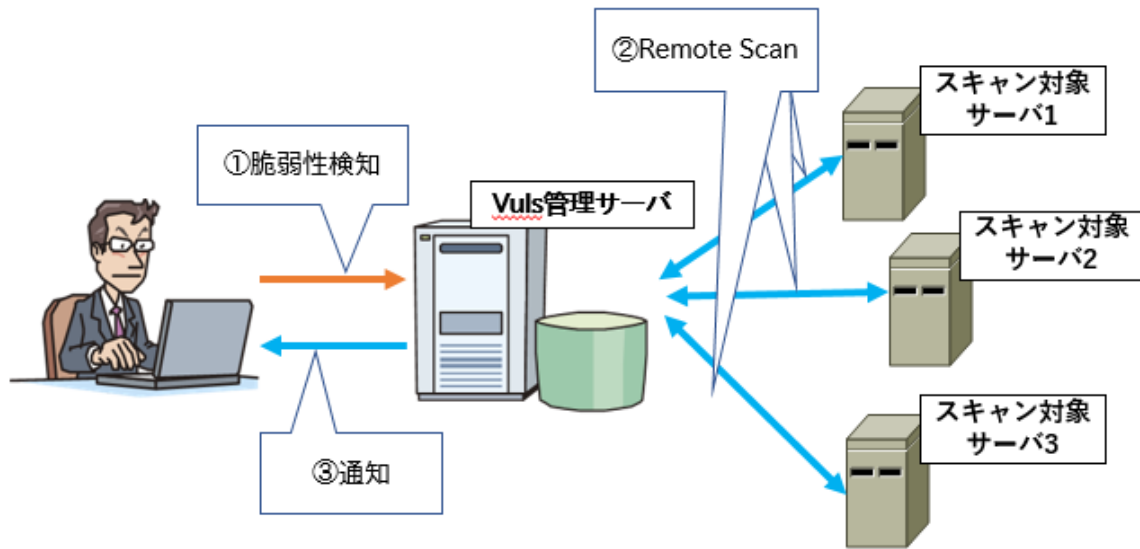


図 4-7-5 Remote Scan Mode の利用イメージ

① 「4.5.手動インストール」または「4.6. (参考) Docker インストール」でセットアップした Vuls 管理サーバを用意する。

② スキャン対象サーバを用意する。

【Vuls 管理サーバ】

③ Vuls 管理サーバ側で公開鍵を生成する。

なお、生成する際に、図 4-7-6 のように公開鍵ファイルの保存場所と SSH 認証するためのパスワード入力求められるのでそれぞれ設定する。

```
$ ssh-keygen -t rsa
```

```
[vulsuser@localhost ~]$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/vulsuser/.ssh/id_rsa):
```

図 4-7-6 公開鍵の生成

- ④ 以下のコマンドで表示された、図 4-7-7 の様な公開鍵情報をメモやクリップボード等で、手元に保管する。

```
$ cat ~/.ssh/id_rsa.pub
```

```
lvulsuser@localhost ~]$ cat ~/.ssh/id_rsa.pub  
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC6sD5hJGV46cNdBb06es2PknpP0nr4a4jfdZFWlZjc  
vu+F1ShHwWim/De3qXyJTB0PC3fkXlgi3YIHbu/FWoihphJQOnEjI9+PrWlCK3P44N3eYDyhyyuQ/HKE  
hrkVs4dPoFoG/ByJGV0FPQzla0b3MfMEJHQS0ZfavISA7M+biNLCHWZnVxQELX7eEEL//our4HtDgxpJ  
oe4E0teUJHwYCOATIVBjiM8L9zxvT2yrp1YPiWsgmjT7nRMuX9uIOBXrghY579sk0hXgD20Pyom+5900  
zBkURHIREfV6nmZTfTAtuwP0xikbSQYJC/3iuYX/rMGuezdp6PGQQdnDl6v vulsuser@localhost.  
localdomain
```

図 4-7-7 公開鍵のコピー内容

【スキャン対象サーバ】

- ⑤ スキャン対象サーバに「scanuser」でログインする。

- ⑥ /home/scanuser 配下に .ssh ディレクトリを作成する。

```
$ mkdir ~/.ssh
```

- ⑦ 「scanuser」に .ssh のフルコントロールを与える。

```
$ chmod 700 ~/.ssh
```

- ⑧ /.ssh 配下に authorized_key の空ファイルを作成する。

```
$ touch ~/.ssh/authorized_keys
```

- ⑨ 「scanuser」に authorized_keys の変更権限を与える。

```
$ chmod 600 ~/.ssh/authorized_keys
```

- ⑩ Vi エディタで authorized_keys を開き、④で保管していた公開鍵情報を貼り付ける。

```
$ vi ~/.ssh/authorized_keys
```

```
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAQAC6sD5hJGV46cNdBb06es2PknpP0nr4a4jfdZFWlZjc  
vu+F1ShHwWim/De3qXyJTB0PC3fkXlgi3YIHbu/FWoihphJQOnEjI9+PrWlCK3P44N3eYDyhyyuQ/HKE  
hrkVs4dPoFoG/ByJGV0FPQzla0b3MfMEJHQS0ZfavISA7M+biNLCHWZnVxQELX7eEEL//our4HtDgxpJ  
oe4E0teUJHwYCOATIVBjiM8L9zxvT2yrp1YPiWsgmjT7nRMuX9uIOBXrghY579sk0hXgD20Pyom+5900  
zBkURHIREfV6nmZTfTAtuwP0xikbSQYJC/3iuYX/rMGuezdp6PGQQdnDl6v vulsuser@localhost.  
localdomain
```

図 4-7-8 公開鍵の貼り付け内容

【Vuls 管理サーバ】

- ⑪ Vuls 管理サーバ側からスキャン対象サーバに対して、以下のリモート(SSH)接続を行い、③で入力したパスワードで「scanuser」にログインする。

```
$ ssh scanuser@172.16.120.160 -i ~/.ssh/id_rsa
```

```
[vulsuser@localhost ~]$ ssh scanuser@172.16.120.160 -i ~/.ssh/id_rsa  
Enter passphrase for key '/home/vulsuser/.ssh/id_rsa':
```

図 4-7-9 公開鍵を使ったログイン

- ⑫ リモート(SSH)接続が確認できたら「scanuser」からログオフし、「vulsuser」でスキャン対象サーバのホスト鍵が登録されていることを確認する。

```
$ cat ~/.ssh/known_hosts
```

```
[vulsuser@localhost ~]$ cat ~/.ssh/known_hosts  
172.16.120.160 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBG1hHMV1kZMOJcPGwjclunx5P4H+/H8eC3xLRjd368T0lBhdUtTzM39KLFYfk0kN4o9G7mi l  
UTVQ9CqDPjCemw=
```

図 4-7-10 スキャン対象サーバのホスト鍵確認

- ⑬ \$HOME に移動する。

```
$ cd $HOME
```

- ⑭ vi エディタで「config.toml」を開き、以下の設定を行う。

```
$ vi config.toml
```

```
[servers]
```

```
[servers.vulsuser]
```

```
host = "172.16.120.160"
```

```
port = "22"
```

```
user = "scanuser"
```

```
keypath = "/home/vulsuser/.ssh/id_rsa"
```

```
[servers]  
  
[servers.localhost]  
host = "localhost"  
port = "local"  
  
[servers.scanuser]  
host = "172.16.120.160"  
port = "22"  
user = "scanuser"  
keypath = "/home/vulsuser/.ssh/id_rsa"
```

図 4-7-11 config.toml 設定内容

⑮ スキャン対象サーバに対して正しく脆弱性検知が行えるか確認する。

(※スキャンサーバの設定を確認する際に、公開鍵のパスワード入力求められる)

```
$ vuls configtest scanuser
```

```
[vulsuser@localhost ~]$ vuls configtest
[Dec 1 00:52:01] INFO [localhost] Validating config...
[Dec 1 00:52:01] INFO [localhost] Detecting Server/Container OS...
[Dec 1 00:52:01] INFO [localhost] Detecting OS of servers...
[Dec 1 00:52:02] INFO [localhost] (1/2) Detected: localhost: centos 7.5.1804
Enter passphrase for key '/home/vulsuser/.ssh/id_rsa':
[Dec 1 00:52:04] INFO [localhost] (2/2) Detected: scanuser: centos 7.5.1804
[Dec 1 00:52:04] INFO [localhost] Detecting OS of containers...
[Dec 1 00:52:04] INFO [localhost] Checking Scan Modes...
[Dec 1 00:52:04] INFO [localhost] Checking dependencies...
[Dec 1 00:52:05] INFO [scanuser] Dependencies ... Pass
[Dec 1 00:52:05] INFO [localhost] Dependencies ... Pass
[Dec 1 00:52:05] INFO [localhost] Checking sudo settings...
[Dec 1 00:52:05] INFO [localhost] Sudo... Pass
[Dec 1 00:52:05] INFO [scanuser] Sudo... Pass
[Dec 1 00:52:05] INFO [localhost] It can be scanned with fast scan mode even i
f warn or err messages are displayed due to lack of dependent packages or sudo s
ettings in fast-root or deep scan mode
[Dec 1 00:52:05] INFO [localhost] Scannable servers are below...
localhost scanuser
```

図 4-7-12 vuls configtest 実行結果

⑯ 脆弱性検知を実行する。

```
$ vuls scan scanuser
```

```
One Line Summary
=====
scanuser      centos7.5.1804  316 installed, 83 updatable    0 exploits
```

図 4-7-13 リモートスキャン実行結果

実行結果の詳細は「4.8.レポート確認 (Vuls 内機能)」参照すること。

4.8. レポート確認(Vuls 内機能)

Vuls が持つ脆弱性検知の機能の紹介と、脆弱性検知を実行して得た結果について以下に示す。

① Vuls Scan

図 4-8-1 の左から、

- ・脆弱性検知を実行したサーバ
- ・サーバの OS 情報
- ・インストールされているソフトウェアの数
- ・ソフトウェアの更新件数
- ・実証コードの公開件数

```
$ vuls scan
```

```
One Line Summary
=====
localhost      centos7.5.1804  366 installed, 21 updatable    0 exploits
```

図 4-8-1 VulsScan 実行結果

Vuls 管理サーバ内には、366 個のソフトウェアがインストールされており、その内、21 個のソフトウェアに更新情報が存在することが分かる。

② One-line summary

図 4-8-2 の左上から、

- ・脆弱性検知を実行したサーバ
- ・脆弱性情報の合計件数と深刻度別件数
- ・脆弱性修正済み件数
- ・インストールされているソフトウェアの件数
- ・ソフトウェアの更新件数
- ・実証コードの公開件数

```
$ vuls report -format-one-line-text
```

```
One Line Summary
=====
localhost      Total: 429 (High:95 Medium:305 Low:29 ?:0)    3/429 Fixed
366 installed, 21 updatable    0 exploits
```

図 4-8-2 One-line summary 実行結果

Vuls 管理サーバ内には、429 件の脆弱性情報 (High が 95 件、Medium が 305 件、Low が 29 件、不明が 0 件) があり、この内、3 件に対して脆弱性対策が実施され、残りの 426 件については未実施であることが分かる。

③ Full report

図 4-8-3 の左上から、

- ・脆弱性検知を実行したサーバ
- ・脆弱性情報の合計件数と深刻度別件数
- ・インストールされているソフトウェアの件数
- ・ソフトウェアの更新件数
- ・実証コードの公開件数
- ・脆弱性毎の詳細情報

```
$ vuls report -format-full-text
```

```
localhost (centos7.5.1804)
=====
Total: 429 (High:95 Medium:305 Low:29 ?:0), 3/429 Fixed, 366 installed, 21 updatable, 0 exploits
-----+-----
| CVE-2016-4658 |
-----+-----
| Max Score | 10.0 HIGH (nvd)
| nvd       | 9.8/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H CRITICAL
| redhat_api | 5.3/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N MODERATE
| jvn       | 9.8/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H CRITICAL
| nvd       | 10.0/AV:N/AC:L/Au:N/C:C/I:C/A:C HIGH
| redhat_api | 4.3/AV:N/AC:M/Au:N/C:P/I:N/A:N MODERATE
| jvn       | 10.0/AV:N/AC:L/Au:N/C:C/I:C/A:C HIGH
| Summary   | xpointer.c in libxml2 before 2.9.5 (as used in Apple iOS before 10, OS X before
```

図 4-8-3 Full report 実行結果

④ View Japanese

③の内容を日本語で確認する

```
$ vuls report -format-full-text -lang ja
```

```
localhost (centos7.5.1804)
=====
Total: 429 (High:95 Medium:305 Low:29 ?:0), 3/429 Fixed, 366 installed, 21 updatable, 0 exploits
-----+-----
| CVE-2016-4658 |
-----+-----
| Max Score | 10.0 HIGH (nvd)
| nvd       | 9.8/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H CRITICAL
| redhat_api | 5.3/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N MODERATE
| jvn       | 9.8/CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H CRITICAL
| nvd       | 10.0/AV:N/AC:L/Au:N/C:C/I:C/A:C HIGH
| redhat_api | 4.3/AV:N/AC:M/Au:N/C:P/I:N/A:N MODERATE
| jvn       | 10.0/AV:N/AC:L/Au:N/C:C/I:C/A:C HIGH
| Summary   | 複数の Apple 製品の libxml2 における任意のコードを実行される脆弱性 複数の
|           | Apple 製品の libxml2 には、任意のコードを実行される、またはサービス運用妨害
|           | (メモリ破損) 状態にされる脆弱性が存在します。
| CWE       | CWE-119: バッファエラー (CWE-119) (nvd)
| CWE       | CWE-416: 解放済みメモリの使用 (CWE-416) (redhat_api)
```

図 4-8-4 View Japanese 実行結果

⑤ TUI

TUI 形式で脆弱性情報を確認する。なお、TUI の操作はキーボードの「TAB キー」と「矢印キー」で操作する。

```
$ vuls tui
```

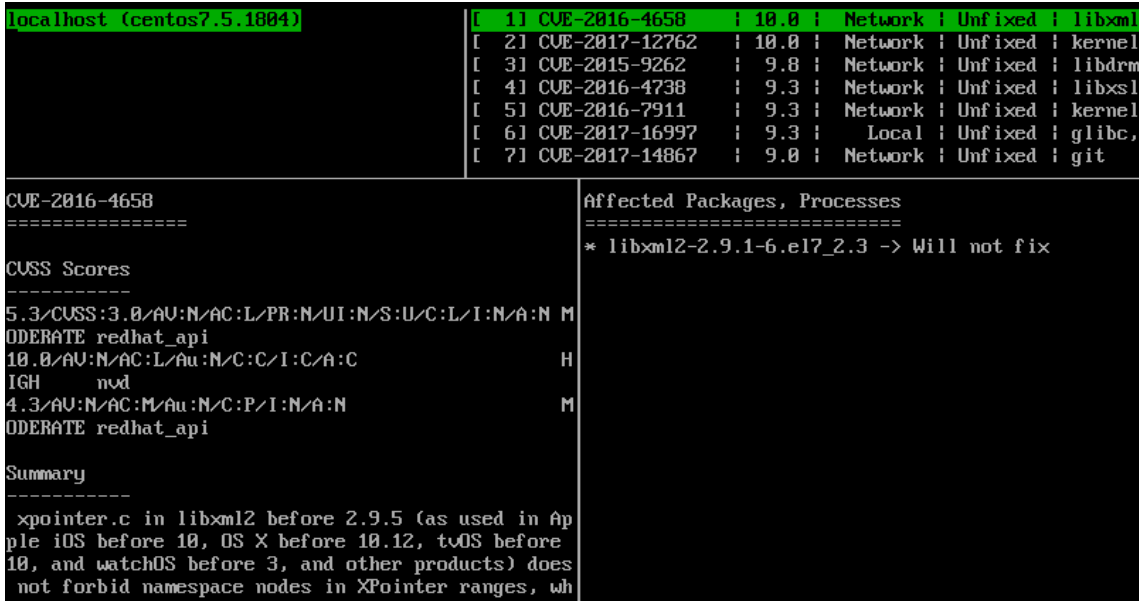


図 4-8-5 TUI 実行結果 1

操作方法例：上から 4 番目にある CVE-2016-4738 の脆弱性情報を確認したい場合。

(1) TAB キーを使い、右上の CVE 情報一覧が選択できるまで押下げる。

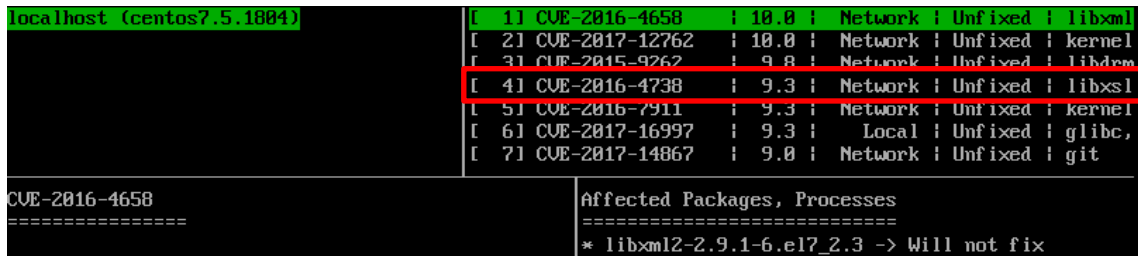


図 4-8-6 TUI 実行結果 2

(2) 矢印キーを使い、上から 4 番目の CVE-2016-4738 を選択する。

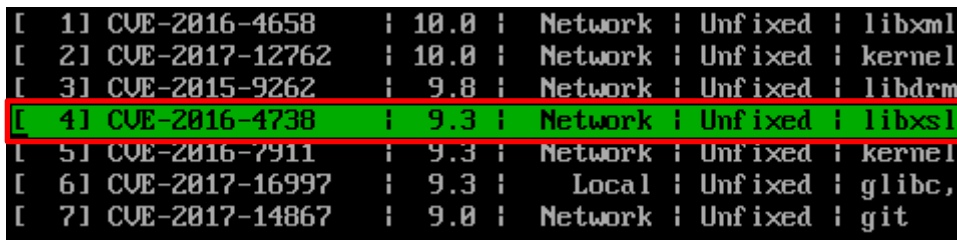


図 4-8-7 TUI 実行結果 3

(3) 再度 TAB キーを使い、左下や右下にある脆弱性関連情報を確認する。

<pre>Summary ----- libxslt in Apple iOS before 10, OS X before 10.12 , tvOS before 10, and watchOS before 3 allows remo te attackers to execute arbitrary code or cause a denial of service (memory corruption) via a crafte d web site. (nvd)</pre>	<pre>Affected Packages, Processes ===== * libxslt-1.1.28-5.el7 -> Will not fix</pre>
--	---

図 4-8-8 TUI 実行結果 4

4.9. VulsRepo

VulsRepo は、脆弱性検知ツール Vuls 用に開発された WebGUI で、「4.7.脆弱性検知」を実行した際に出力されたファイル (.json) を VulsRepo 内で読み込み、レポートとしてブラウザ上に表示させる。

このレポートには、脆弱性情報の検知結果を基として、深刻度別の脆弱性情報を色つきのマトリックス表で表示する機能、脆弱性が確認されたソフトウェアに紐づく CVE とその概要を一覧で表示する機能、詳細な脆弱性情報の CVSS の評価項目別にレーダーチャート表示する機能等、利用者が確認し易い形式で検知した脆弱性情報を可視化、把握することが可能である。

4.9.1. VulsRepo インストール

VulsRepo をインストールする。

<注意事項：2018 年 11 月時点>

- ・ Vuls のバージョンは 0.4.0 以上であること
- ・ ウェブブラウザは “Google Chrome” か “Firefox” を使用すること

① \$HOME に移動する

```
$ cd $HOME
```

② git で vulsrepo.git をコピーする

```
$ git clone https://github.com/usiusi360/vulsrepo.git
```

③ \$HOME/vulsrepo/server に移動する

```
$ cd $HOME/vulsrepo/server
```

④ vulsrepo-config.toml.sample の内容を vulsrepo-config.toml にコピーする

```
$ cp vulsrepo-config.toml.sample vulsrepo-config.toml
```

⑤ vi エディタで vulsrepo-config.toml を開き、以下の内容を設定する

```
$ vi vulsrepo-config.toml
```

```
[Server]
```

```
rootPath = "/home/vulsuser/vulsrepo"
```

```
resultsPath = "/home/vulsuser/results"
```

```
serverPort = "5111"
```

```
[Server]
rootPath = "/home/vulsuser/vulsrepo"
resultsPath = "/home/vulsuser/results"
serverPort = "5111"
```

図 4-9-1 vulsrepo-config.toml 設定内容

- ⑥ VulsRepo で使うサーバーポート「5111」を恒常的に開く設定をする。

```
$ sudo firewall-cmd --add-port=5111/tcp --zone=public --permanent
```

- ⑦ ⑥で設定した内容を反映させる。

```
$ sudo firewall-cmd --reload
```

- ⑧ 設定した内容が正しく反映されているか確認する。

```
$ sudo firewall-cmd --list-all
```

```
[vulsuser@localhost server]$ sudo firewall-cmd --list-all
[sudo] password for vulsuser:
public (active)
  target: default
  icmp-block-inversion: no
  interfaces: ens33
  sources:
  services: ssh dhcpv6-client http
  ports: 8089/tcp 5111/tcp
  protocols:
  masquerade: no
  forward-ports:
  source-ports:
  icmp-blocks:
  rich rules:
```

図 4-9-2 firewall-cmd 設定内容

4.9.2. VulsRepo 実行

VulsRepo を使い、ブラウザ上にレポートを表示させる。

- ① Vuls Scan を実行する

```
$ vuls scan
```

- ② ①の実行結果を JSON ファイルで出力する

```
$ vuls report -format-json
```

- ③ ②で出力された JSON ファイルが /home/vulsuser/results/ 配下に正しく出力されているか確認する

```
$ cd /home/vulsuser/results/  
$ ls
```

```
[vulsuser@localhost results]$ cd /home/vulsuser/results/  
[vulsuser@localhost results]$ ls  
2018-11-30T11:03:25+09:00 current
```

図 4-9-3 results 確認

- ④ \$HOME/vulsrepo/server に移動する

```
$ cd $HOME/vulsrepo/server
```

- ⑤ vulsrepo-server を起動する

```
$ ./vulsrepo-server
```

- ⑥ ブラウザを起動して、URL 欄に Vuls 管理サーバの IP アドレスとで「4.9.1. VulsRepo インストール」で設定したポート番号を指定し、アクセスする。

```
http://172.16.120.150:5111
```



図 4-9-4 VulsRepo の URL

- ⑦ 以下の画面が表示されたら、赤枠のリストボタンをクリックする

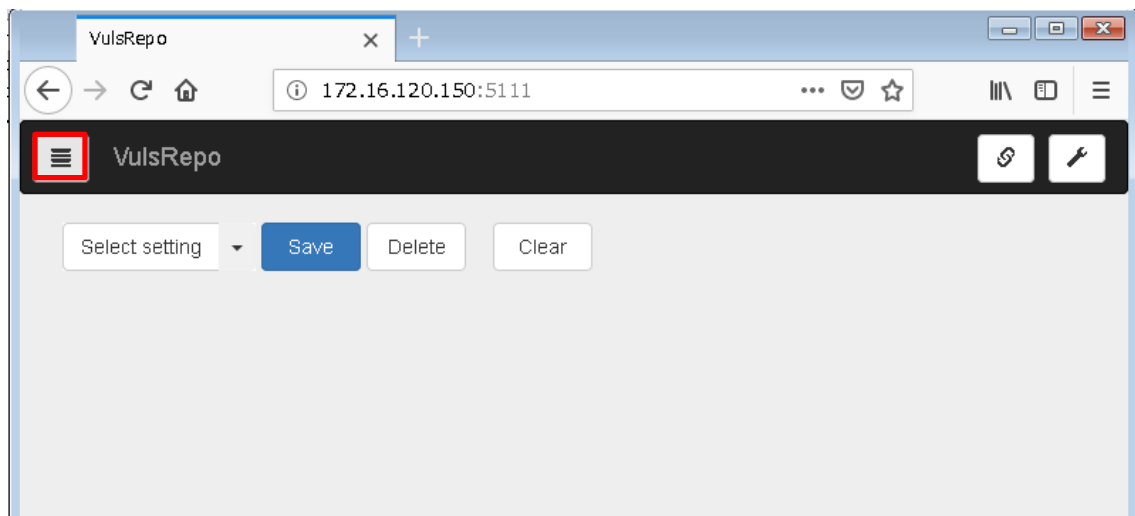


図 4-9-5 VulsRepo 初期画面

- ⑧ ⑦まで実行すると、これまでサーバ毎に脆弱性検知を実行した結果である JSON ファイルが一覧表示される。その中から、レポートで確認したいファイルを選択し、Submit ボタンを押下げる。

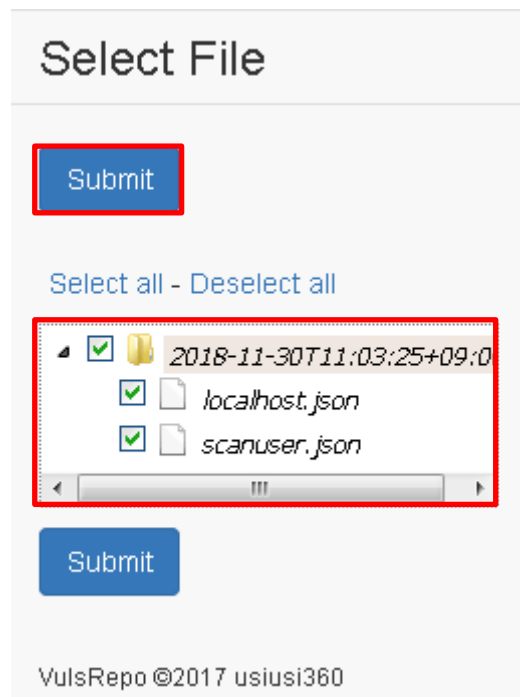


図 4-9-6 VulsRepo ファイル選択画面

- ⑨ Vuls 管理サーバ (Localhost) とスキャン対象サーバ (scanuser) の脆弱性検知結果を選択すると、図 4-9-7 のように表示される。

ScanTime	ServerName	Container	CVSS Severity		
			CVSS Score	1.5	1.9
2018-11-30T23:27:12+09:00	localhost	None	1	6	37
	scanuser	None	1	6	33
Totals			1	12	70

図 4-9-7 Vulsrepo 確認画面

4.9.3. VulnsRepo 利用

VulnsRepo で確認できるレポートの中から、サーバの状況をより把握しやすくするための表示例を以下に示す。

(1) 脆弱性検知を行ったサーバに含まれる脆弱性の深刻度別件数

- ① 図 4-9-8 のように、「02. Graph: CVSS-Severity => CVSS-Score」を選択する。

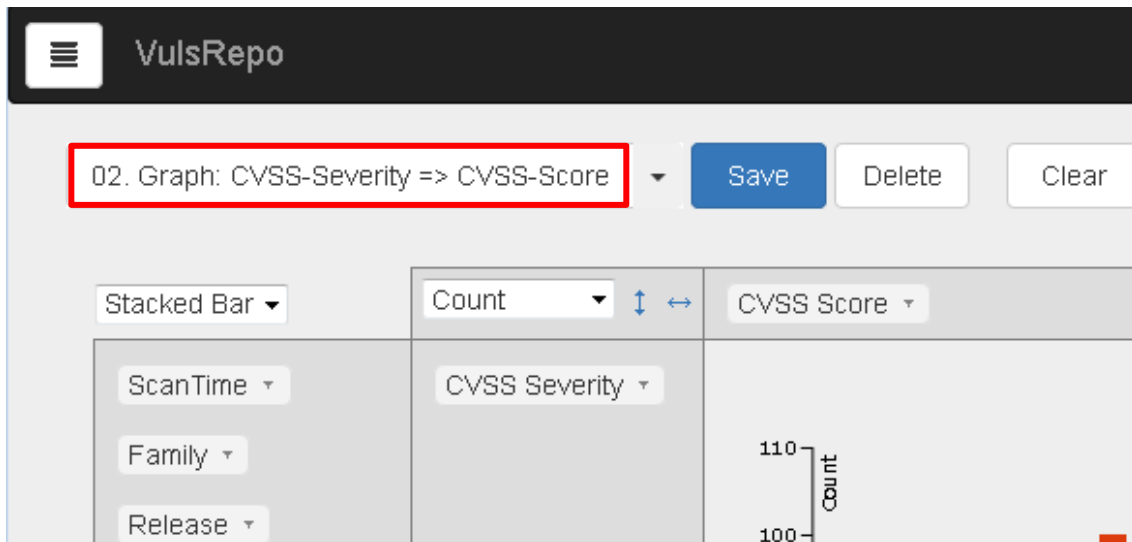


図 4-9-8 Vulnsrepo 表示例 1

- ② 以下のグラフが表示される。

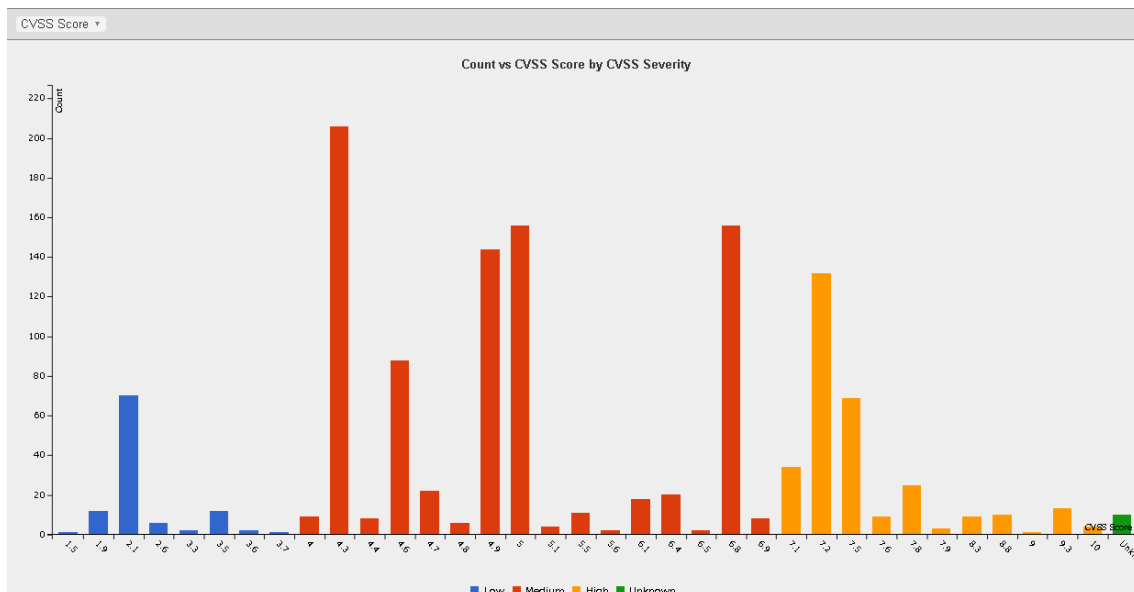


図 4-9-9 vulnsrepo 表示例 1

検証環境（Vuls 管理サーバとスキャン対象サーバ）の検知結果を、深刻度別件数のグラフで見ると、Medium（4.3）が最も件数が多く、また、High（7.1）以上となっている危険性が高い脆弱性情報を計 147 件確認した。

(2) 脆弱性対策の対応状況の確認

① 図 4-9-10 のように、「Select setting」、「Count」を選択した状態で、縦列に「CVSS Severity」と「CVSS Score」を、横列に「ServerName」「NotFixedYet」をドラッグ&ドロップで配置する。

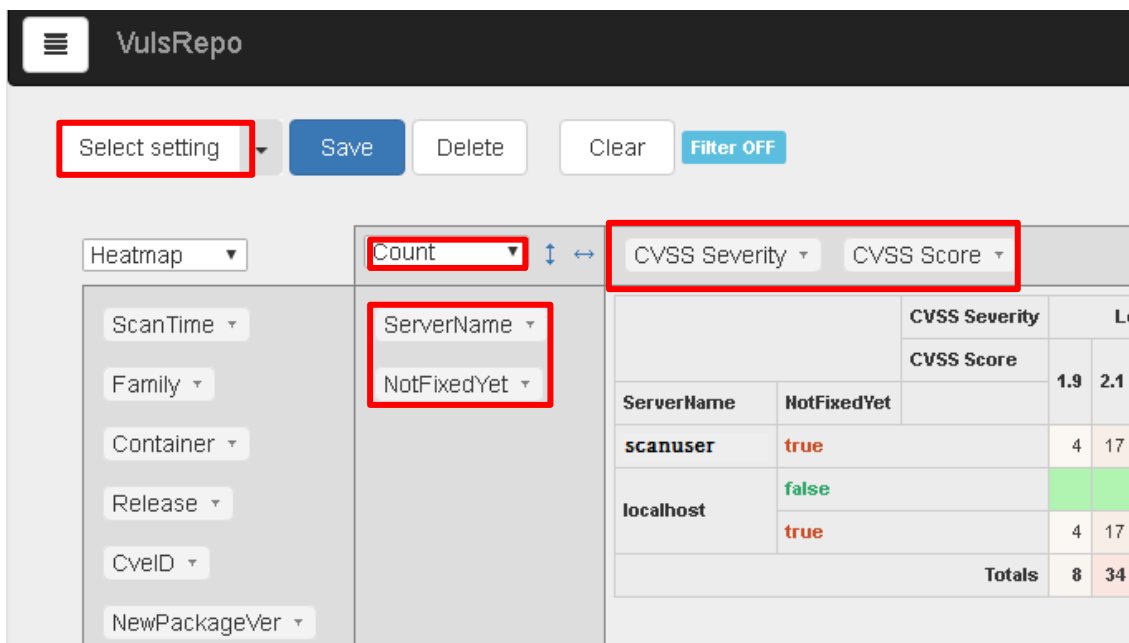


図 4-9-10 vulsrepo 表示例 2

② 以下の表が確認できる。

ServerName	NotFixedYet	CVSS Severity	Low							High							Unknown	Totals		
		CVSS Score	1.5	1.9	2.1	2.6	3.3	3.5	3.6	3.7	7.6	7.8	7.9	8.3	8.8	9	9.3		10	Unknown
localhost	false										2									
	true			6	37	3	1	5			5	11		5	6	1	7	2	5	
scanuser	false		1					3			1		2	3				1		
	true			6	33	3	1	4			4	10		4	4		5	2	5	
Totals			1	12	70	6	2	12	2	1	9	25	3	9	10	1	13	4	10	1,285

図 4-9-11 vulsrepo 表示例 2（図中省略あり）

検証環境では、Localhost（Vuls 管理サーバ）側に 364 件の脆弱性、scanuser（スキャン対象サーバ）側は 340 件の脆弱性を検知しており、その内、Localhost 側においては 10 件の脆弱性対策を、scanuser 側に 103 の脆弱性対策を実施していることを確認した。

(3) 脆弱性検知で確認した脆弱性情報の一覧 (CVSS Score 降順)

① 図 4-9-12 のように、「Maximum」 - 「CVSS Score」 - 「↑」を選択した状態で、横列に「ServerName」「CveID」「CVSS Severity」「CVSS Score」「Summary」を配置する。(縦列は何もしない。)

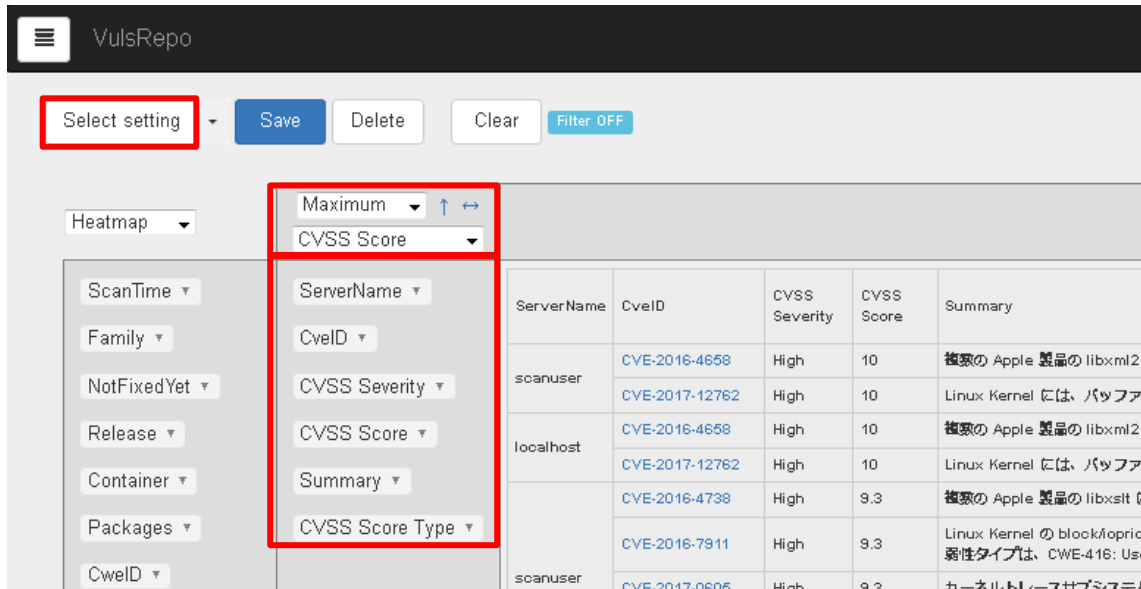


図 4-9-12 vulsrepo 表示例 3

② 以下の表が確認できる。

なお、図 4-9-13 は、Localhost (Vuls 管理サーバ) の検知結果ファイルのみを選択した状態で、一覧表示した結果である。

ServerName	CVSS Score	CveID	CVSS Severity	Summary	Totals
	10	CVE-2016-4658	High	複数の Apple 製品の libxml2 には、任意のコードを実行される、またはサービス運用妨害 (メモリ破壊) 状態にされる脆弱性が存在します。	10.00
		CVE-2017-12762	High	Linux Kernel には、バッファエラーの脆弱性が存在します。	10.00
	9.3	CVE-2016-4738	High	複数の Apple 製品の libxslt には、任意のコードを実行される、またはサービス運用妨害 (メモリ破壊) 状態にされる脆弱性が存在します。	9.30
		CVE-2016-7911	High	Linux Kernel の block/proprio.o の get_task_ioprio 関数には、競合状態により、権限を取得される、またはサービス運用妨害 (解放済みメモリの使用 (use-after-free)) 状態にされる脆弱性が存在します。補足情報: CWE による脆弱性タイプは、CWE-416: Use After Free (解放済みメモリの使用) と識別されています。 https://cwe.mitre.org/data/definitions/416.html	9.30
		CVE-2017-0605	High	カーネルトレースサブシステムには、権限を昇格される脆弱性が存在します。本脆弱性は、Android ID: A-35399704 および Qualcomm QC-CR#1048480 として公開されています。	9.30
		CVE-2017-16997	High	GNU C Library (別名 glibc または libc6) には、信頼性のない検索パスに関する脆弱性が存在します。	9.30
	9	CVE-2017-14867	High	Git には、入力確認に関する脆弱性が存在します。	9.00
				NetworkManager is a system network service that manages network devices and connections, attempting to keep active network	8.80

図 4-9-13 vulsrepo 表示例 3

CVSS Score の降順を指定したことで、深刻度の高い脆弱性を上から順に確認することができる。さらに、CveID 列のリンクをクリックすることで、図 4-9-14 のように、その脆弱性についてより詳しい情報を確認することも可能である。

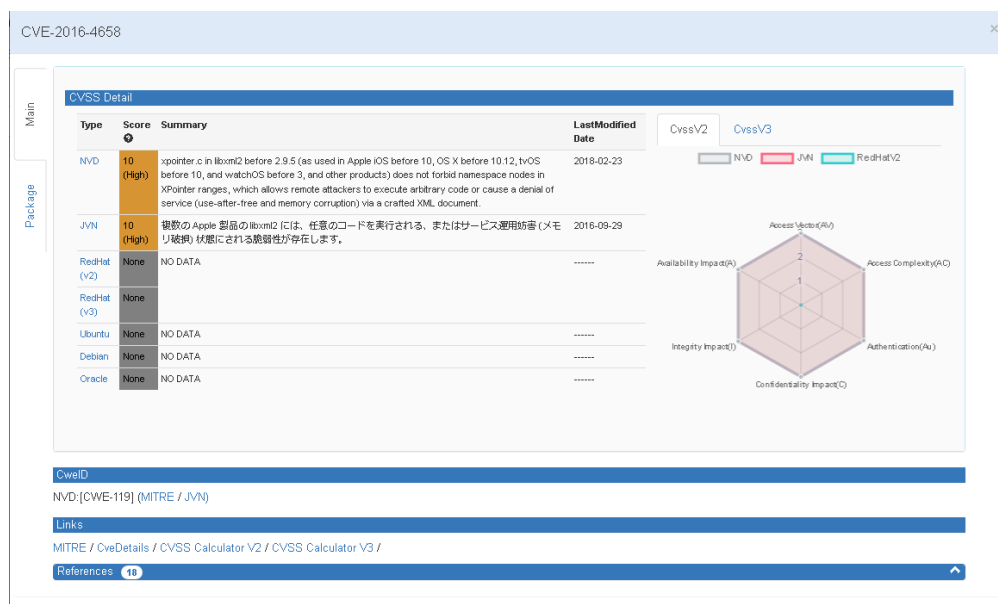


図 4-9-14 vulsrepo 表示例 4

4.10.cron 設定

脆弱性データベース（NVD や JVN iPedia）に登録される脆弱性情報は日々更新されるため、本書では OS の機能である cron を使用して、定期的にデータベースを確認するようにし、情報が更新されれば go-cve-dictionary 内に取り込んでいく設定をした。さらに、脆弱性検知を定期的に行わせ、メールでシステム管理者に通知する cron の設定についても以下の通り行った。

- ① 「Remote Scan Mode」で定期的な収集作業と脆弱性検知を行う場合は、スキャン対象サーバに予めリモート接続しておく。

【Vuls 管理サーバ】

```
$ ssh scanuser@172.16.120.160 -i ~/.ssh/id_rsa
$ exit
```

【スキャン対象サーバ】

```
$ vuls configtest scanuser
```

- ② Vuls 管理サーバ上の「vulsuser」で、cron を新規作成する。
 (※RedHat 系の Linux では、/etc/crontab を直接編集することは推奨されていないため、定期的に行いたいコマンドについては「cron.monthly」、「cron.weekly」、「cron.daily」、「cron.hourly」配下のファイル等に設定を行うようにする。)

```
$ crontab -e
```

- 毎週月・火・水・木・金の 00 : 00 時に go-cve-dictionary を自動更新する

```
0 0 ** 1,2,3,4,5 sudo -u vulsuser /home/vulsuser/go/bin/go-cve-dictionary  
fetchjvn -latest -dbpath=/home/vulsuser/cve.sqlite3
```

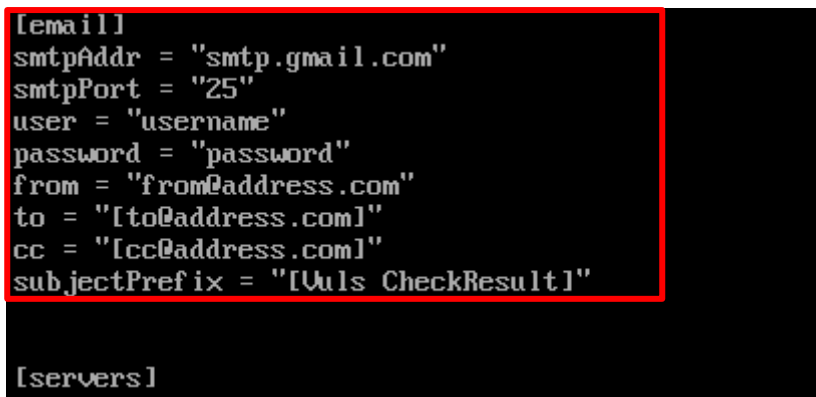
- 毎週月・火・水・木・金の 00 : 00 時に vuls scan を自動実行する

```
0 0 ** 1,2,3,4,5 sudo -u vulsuser /home/vulsuser/go/bin/vuls scan -  
config=/home/vulsuser/config.toml -results-dir=/home/vulsuser/results
```

- 毎週月・火・水・木・金の 09 : 00 時に CVSS が High 以上の情報を指定したメールアドレスに送信する。

③ vi エディタで「config.toml」を開き、以下の設定を行う。(※実際の設定内容ではない)

```
[email]  
smtpAddr = "smtp.gmail.com"  
smtpPort = "25"  
user = "username"  
password = "password"  
from = "from@address.com"  
to = "[to@address.com]"  
cc = "[cc@address.com]"  
subjectPrefix = "[Vuls CheckResult]"
```



```
[email]  
smtpAddr = "smtp.gmail.com"  
smtpPort = "25"  
user = "username"  
password = "password"  
from = "from@address.com"  
to = "[to@address.com]"  
cc = "[cc@address.com]"  
subjectPrefix = "[Vuls CheckResult]"  
  
[servers]
```

図 4-10-1 config.toml 設定内容

```
0 9 ** 1,2,3,4,5 sudo -u vulsuser /home/vulsuser/go/bin/vuls report -cvedb-  
path=/home/vulsuser/cve.sqlite3 -ovaldb-path=/home/vulsuser/oval.sqlite3 -  
config=/home/vulsuser/config.toml -results-dir=/home/vulsuser/results -format-json -to-  
email -format-full-text -cvss-over=7 -lang=ja
```

④ ③で設定した内容が正しく実行できているか確認する。

```
$ vi /var/spool/mail/vulsuser
```

```
From vulsuser@localhost.localdomain Sun Nov 18 21:50:02 2018
Return-Path: <vulsuser@localhost.localdomain>
X-Original-To: vulsuser
Delivered-To: vulsuser@localhost.localdomain
Received: by localhost.localdomain (Postfix, from userid 1000)
        id 7EDE720767EB; Sun, 18 Nov 2018 21:50:02 +0900 (JST)
From: "(Cron Daemon)" <vulsuser@localhost.localdomain>
To: vulsuser@localhost.localdomain
Subject: Cron <vulsuser@localhost> sudo -u vulsuser /home/vulsuser/go/bin/go-cve-dictionary fetch
Content-Type: text/plain; charset=UTF-8
Auto-Submitted: auto-generated
Precedence: bulk
X-Cron-Env: <XDG_SESSION_ID=20>
X-Cron-Env: <XDG_RUNTIME_DIR=/run/user/1000>
X-Cron-Env: <LANG=ja_JP.UTF-8>
X-Cron-Env: <SHELL=/bin/sh>
X-Cron-Env: <HOME=/home/vulsuser>
X-Cron-Env: <PATH=/usr/bin:/bin>
X-Cron-Env: <LOGNAME=vulsuser>
X-Cron-Env: <USER=vulsuser>
Message-Id: <20181118125002.7EDE720767EB@localhost.localdomain>
Date: Sun, 18 Nov 2018 21:50:02 +0900 (JST)

t=2018-11-18T21:50:01+0900 lvl=info msg="Fetching... https://jvndb.jvn.jp/ja/feed/checksum.txt"
t=2018-11-18T21:50:02+0900 lvl=info msg="Fetched.. https://jvndb.jvn.jp/ja/feed/checksum.txt"
t=2018-11-18T21:50:02+0900 lvl=info msg="up to date: https://jvndb.jvn.jp/ja/rss/jvndb.rdf"
t=2018-11-18T21:50:02+0900 lvl=info msg="up to date: https://jvndb.jvn.jp/ja/rss/jvndb_new.rdf"
t=2018-11-18T21:50:02+0900 lvl=info msg="Already up to date"
```

図 4-10-2 go-cve-dictionary 自動更新結果

```
time="Nov 18 21:50:01" level=info msg="Start scanning"
time="Nov 18 21:50:01" level=info msg="config: /home/vulsuser/config.toml"
time="Nov 18 21:50:01" level=info msg="validating config..."
time="Nov 18 21:50:01" level=info msg="Detecting Server/Container OS... "
time="Nov 18 21:50:01" level=info msg="Detecting OS of servers... "
time="Nov 18 21:50:02" level=info msg="(1/1) Detected: localhost: centos 7.5.1804"
time="Nov 18 21:50:02" level=info msg="Detecting OS of containers... "
time="Nov 18 21:50:02" level=info msg="Checking Scan Modes... "
time="Nov 18 21:50:02" level=info msg="Detecting Platforms... "
time="Nov 18 21:50:02" level=info msg="(1/1) localhost is running on other"
time="Nov 18 21:50:02" level=info msg="Scanning vulnerabilities..."
time="Nov 18 21:50:02" level=info msg="Scanning vulnerable OS packages..."
time="Nov 18 21:50:02" level=info msg="Scanning in fast mode"

One Line Summary
=====
localhost          centos7.5.1804  361 installed, 0 updatable      0 exploits
```

図 4-10-3 Vuls scan 自動更新結果

5. 検証結果

Vuls の動作検証の結果について以下に示す。

(1) 手動と比べ脆弱性関連情報を格段に早く収集可能

本書の動作環境では、Vuls を利用することで、サーバ内にインストールされているソフトウェアの脆弱性有無をたった数分～数 10 分程度で網羅的に確認できた。これにより、手動による収集作業で使っていた多くの時間を脆弱性の影響調査に回す等、脆弱性対策における作業の効率化を期待できると考える。また、VulsRepo も並行して活用することで、サーバ内の状況が視覚的に把握しやすくなり、脆弱性が含まれるソフトウェアの調査等をより効果的に行うことも可能となる。これまで、サーバ内に存在する脆弱性情報の収集作業を手動で行い、負担を感じていたシステム管理者は、本ツールの利用を検討してみたい。

なお、Vuls の利用を検討するにあたり、自組織でプロキシサーバを使っている場合は注意が必要である。本書の検証環境では、プロキシサーバが存在する前提で検証環境を作っていたが、一部のコマンドはオプションとしてプロキシを指定しないとデータの取得ができないことがあった。そのため、コマンドを打つ際は `-h` (ヘルプ) で確認したり、開発者ページを参照することが望ましい。

(2) 短期間で機能拡張が行われる反面、バージョンアップを行う手間が掛かる

Vuls は、本書の動作検証の間に数回程、機能追加やバグ修正のためのソフトウェア更新が行われた。Vuls で利用するソフトウェアのバージョンを上げるには、その都度、再インストールが必要となるため、Vuls の利用者はこの点を含めた運用手順を考慮しておく必要がある。

(3) Vuls を運用する際の活用イメージ

図 5-1 は、Vuls を利用するにあたり、システム管理者が容易に構築・運用することが可能と思われる利用イメージを示した。本書の 4 章で行った検証内容が参考として使えるため、Vuls を初めて利用するシステム管理者は、これらの機能が使えるよう構築してみたい。

① cron を使った定期的なデータベース更新

日々更新される脆弱性データベースの脆弱性情報を定期的に自動収集する。
→「4.10.cron 設定」参照すること。

② cron を使った定期的な脆弱性検知

脆弱性検知を自動化し、検知結果をログとして保管する。
→「4.10.cron 設定」参照すること。

③ メールで利用者に通知

②の実行結果から影響度の高い脆弱性情報のみを、利用者にメールで通知する。
→「4.10.cron 設定」参照すること。

④ VulsRepo で確認

③で受信した脆弱性情報を VulsRepo を使って確認。適切な脆弱性対策を実施する。
→「4.9.VulsRepo」参照すること。

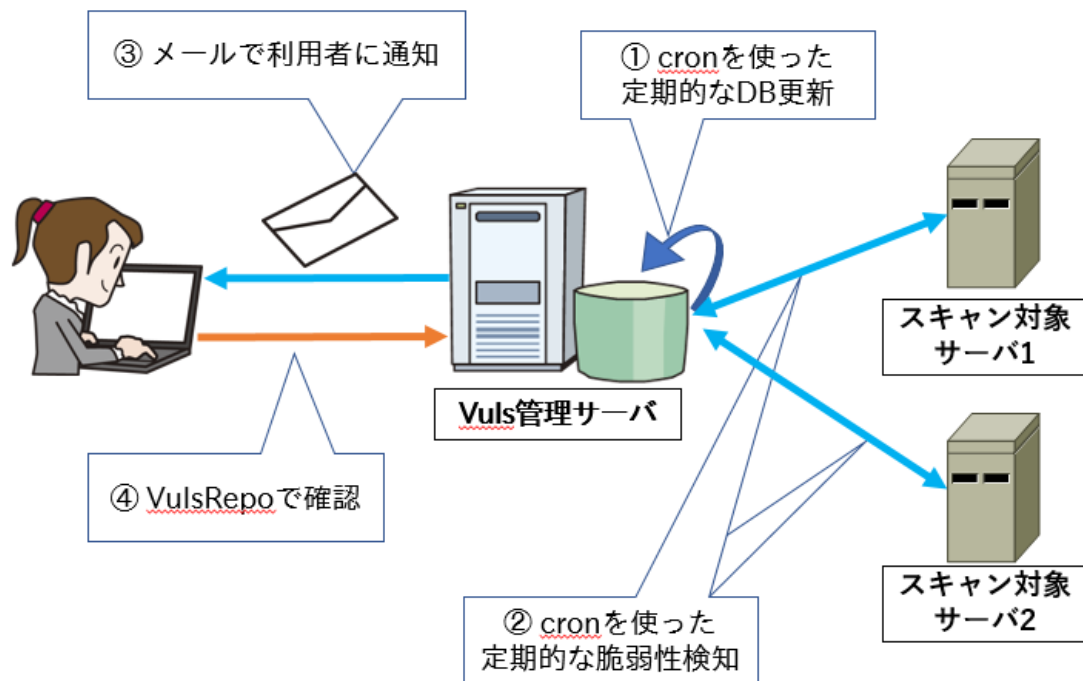


図 5-1 Vuls 活用例

おわりに

本書では、脆弱性を悪用する攻撃から自組織で管理するシステムを守るために、システム管理者がどのように脆弱性対策を進めていくべきかを整理し、解説を行った。この資料を参考に、脆弱性対策で実施する作業を組織内に定常化させ、脆弱性対策をより適切に進められることを期待する。なお、組織内で利用しているソフトウェアをシステム管理者が網羅的に把握、管理することは困難であるため、脆弱性対策の支援ツールを導入したり、必要に応じてシステムの開発ベンダに協力を依頼したりすることも検討して欲しい。

また、本書後半では、脆弱性検知ツール Vuls の動作検証を行い、その検証結果について示した。このツールを使い脆弱性関連情報の収集作業を自動化することで、システム管理者に掛かる作業工数の低減が期待できると考えている。なお、検証結果で触れている通り、構築する環境によっては手順を幾らか変える必要があるため、本番環境に導入する前に仮想環境上で試しておく等、十分な検証を行ってから導入を推奨する。

今回のテクニカルウォッチが、システム管理者および組織全体の脆弱性対策の促進に役立てれば幸いである。

IPA テクニカルウォッチ

「脆弱性対策の効果的な進め方（ツール活用編） ～ 脆弱性検知ツール Vuls を利用した脆弱性対策 ～」

[発行] 2019年2月21日

[著作・制作] 独立行政法人情報処理推進機構 セキュリティセンター

[執筆者] 渡邊 祥樹 大友 更紗 亀山 友彦