Melhorias no Processo de Identificação de Componentes Vulneráveis: Decidindo Sobre Atualizações

Improvements to the Identification Process of Vulnerable Components: Deciding About Updates

Bruna Vuicik Mocelin
Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, Brasil
bvuicik@gmail.com

Kleinner Farias
Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, Brasil
kleinnerfarias@unisinos.br

Lucian Gonçales
Universidade do Vale do Rio dos Sinos São Leopoldo, RS, Brasil
lucianj@edu.unisinos.br

Vinicius Bischoff
Universidade do Vale do Rio dos Sinos
São Leopoldo, RS, Brasil
viniciusbischof@edu.unisinos.br

ABSTRACT

Applications [1] may contain vulnerabilities for a variety of reasons, one of which is the use of vulnerable components. One of the solutions adopted to eliminate the vulnerabilities inserted by such components is to update the component to a more recent version that corrects the vulnerability. However, updating a component may require code refactoring, updating other components and inserting new vulnerabilities in the application. There are several tools that perform the analysis and management of dependencies of the projects, but few tools present information about vulnerabilities of the new versions, incompatibilities and updates of the dependencies of the components. This article, therefore, presents dep|ct (depict), a tool that aims to identify the known vulnerable components used by the applications and help in the decision on the updating of such components, in order to mitigate the vulnerabilities added to the projects through the vulnerable dependencies. Results of the empirical evaluation carried out on two projects show that the tool can be used to assist in deciding on the update of known vulnerable components.

CCS CONCEPTS

• **Security and privacy** → Software and application security → Software security engineering.

KEYWORDS

Known Vulnerable Components; Components Update; Applications Security; Vulnerabilities.

ACM Reference format:

B. V. Mocelin, K. Farias, L. Gonçales, V. Bischoff. 2018. SIG Proceedings Paper in word Format. In *Proceedings of Simpósio Brasileiro de Sistemas de Informação, Caxias do Sul, Rio Grande do Sul, Brazil, June 2018 (SBSI'18),* 8 pages. https://doi.org/10.1145/123 4

[1]

SBSI 2018, June 2018, Caxias do Sul, Rio Grande do Sul, Brazil.

https://doi.org/10.1145/123_4

1 INTRODUÇÃO

Uma vulnerabilidade é um defeito no projeto, implementação, controles internos ou procedimentos de segurança de um sistema aplicação que, quando explorado, pode resultar em uma violação de segurança [16]. Aplicações corporativa podem conter vulnerabilidades por vários motivos, incluindo alteração inapropriada do código da aplicação, falhas na especificação, defeitos de implementação e pressuposições incorretas sobre as entradas recebidas [1,2]. Os componentes vulneráveis conhecidos são aqueles que possuem alguma vulnerabilidade documenta e divulgada em meios públicos, como ferramentas de gerenciamento de defeitos e bancos de dados de vulnerabilidades [2]. A utilização de tais componentes aumenta o risco da ocorrência de violações de segurança, pois uma vez que a vulnerabilidade é conhecida explorá-la se torna mais fácil, podendo envolver o uso de sequências de instruções específicas ou ferramentas automatizadas [7].

Em uma tentativa de dar suporte à comunidade sobre segurança em aplicações, a *Open Web Application Security Project* (OWASP) publica periodicamente o relatório OWASP Top 10 com os 10 riscos de segurança mais críticos encontrados em aplicações web. A utilização de componentes vulneráveis conhecidos foi introduzida na nona posição do último relatório, publicado em 2013 0. Ocorrências de grande repercussão também aumentam a atenção para a utilização de tais componentes, como a divulgação em 2014 da vulnerabilidade *Heartbleed* na biblioteca

de criptografia OpenSSL 0. Informações, atividades e diretrizes relacionadas ao controle da utilização de componentes de terceiros vêm sendo incorporadas a padrões e organizações internacionais, incluindo o Padrão de Segurança de Dados da Indústria de Cartões de Pagamento 0 e a *Financial Services Information Sharing and Analysis Center* 0. Já existem aplicações que auxiliam na identificação da utilização de componentes vulneráveis conhecidos. Uma das ações para solucionar o problema é atualizar o componente vulnerável para uma versão mais recente, que corrige a vulnerabilidade, porém, a atualização do componente pode envolver outras questões não previstas e explicitadas pelas ferramentas atuais, incluindo refatoração de código, propagação de atualização de outros componentes e verificação de novas vulnerabilidades inseridas na aplicação através da atualização 0.

Portanto, este artigo tem como objetivo auxiliar na decisão sobre a atualização de componentes vulneráveis conhecidos, sendo tal atualização planejada com o propósito de mitigar as vulnerabilidades adicionadas aos projetos através das dependências vulneráveis. Para alcançar tal objetivo, foi realizada primeiramente uma comparação entre ferramentas atuais de detecção de componentes vulneráveis conhecidos. Essa comparação permitiu identificar as lacunas existentes. Desse modo, este trabalho propõe a *dep|ct* (lê-se depict), uma ferramenta para identificar tais componentes e auxiliar na decisão sobre sua atualização.

O restante do artigo está organizado da seguinte forma. Seção 2 apresenta descreve os trabalhos relacionados. Seção 3 apresenta a ferramenta proposta. Seção 4 descreve os resultados da avaliação empírica. Por fim, a Seção 5 apresenta as conclusões.

2 TRABALHOS RELACIONADOS

Esta Seção descreve os trabalhos contemporâneos em relação ao tema dessa pesquisa: Suporte a decisão sobre atualizações de aplicações que contém "componentes vulneráveis conhecidos". Compreende-se que "componentes vulneráveis conhecidos" são artefatos na aplicação que contem vulnerabilidades listadas e catalogadas em um banco de dados digital, tais como, o *National Vulnerability Database* (NVD) 0. Desse modo, a Seção 2.1 apresenta as ferramentas que realizam a análise das dependências de projetos. A Seção 2.2 descreve os principais requisitos que a literatura sugere para que uma ferramenta de detecção de vulnerabilidades seja apropriada para suporte a decisão de atualizações. Finalmente, a Seção 2.3 apresenta uma análise comparativa entre as ferramentas, em relação aos requisitos encontrados na literatura.

2.1 Ferramentas

Há várias ferramentas disponíveis que visam a análise e o gerenciamento das dependências de projetos com relação a componentes vulneráveis conhecidos na literatura e na indústria. Abaixo as principais delas são descritas:

Sonartype Application Health Check 0: ferramenta que permite a verificação de binários de aplicações Java em vários formatos tais como .war, .jar, .zip, .tar, e .gz. As vulnerabilidades são obtidas de fontes públicas e proprietárias. O resultado da análise é enviado por e-mail.

bitHound [3]: analisa arquivos *JavaScript*, *TypeScript* e JSX, módulos *npm* e componentes do *bower*. As vulnerabilidades das dependências são verificadas através do *Node Security Project*. Possui integração com repositórios *Git* hospedados no GitHub e no *BitBucket*, com os softwares de integração entre equipes Slack e *HipChat* e plataforma de integração contínua *Codeship*.

Black Duck Hub [4]: suporta a identificação de dependências em projetos Java, C++, C# – dentre outros – e obtém as vulnerabilidades dos bancos de dados NVD e VulnDB. Além de listar as vulnerabilidades das versões dos componentes incluídas no projeto, permite pesquisar as demais versões e suas vulnerabilidades. Permite integração com a ferramenta de integração contínua Jenkins.

Bundler-audit [5]: verifica as vulnerabilidades das dependências de aplicações Ruby. O resultado da análise lista as vulnerabilidades encontradas e possíveis soluções, como a atualização para outra versão. Utiliza o banco de dados ruby-advisory-db, que obtém dados através do banco de dados OSVDB.

Codenomicon AppCheck [9]: utiliza o arquivo executável (.msi, .exe, .dmg, .dpkg, .jar, dentre outros) para extrair as informações de dependências da aplicação. As vulnerabilidades são obtidas através de várias fontes de dados, incluindo o banco de dados NVD. Para permitir a análise automática de projetos a ferramenta disponibiliza API para submissão e obtenção do resultado da análise.

Contrast [10]: ferramenta que monitora as vulnerabilidades em tempo de execução da aplicação. Possui suporte às linguagens Java, .NET, Node.js, C# e Visual Basic e permite integração com o IDE Eclipse para auxiliar na identificação de vulnerabilidades como SQL *Injection* e *Cross-Site Scripting*.

Gemnasium [14]: suporta projetos que utilizam gerenciamento de dependências (gems, npm, pypi, packagist e bower). Esta ferramenta verifica o conjunto de dependências compatíveis. Permite a atualização automática do componente, e receber notificações por e-mail ou através dos softwares de integração, tais como Slack. As vulnerabilidades são obtidas a partir de várias fontes, tais como as listas de e-mail oss-security,

Nexus Lifecycle [28]: possui todas as funcionalidades da ferramenta *Application Health Check* [27]. Porem suporta a integração com ferramentas utilizadas no processo de software, como Eclipse, Maven, Atlassian Bamboo, Jenkins, dentre outras.

Node Security Project [20]: este projeto disponibiliza informações sobre vulnerabilidades nos módulos Node.js. O resultado da análise apresenta a versão mínima que corrige a vulnerabilidade (quando disponível).

OWASP Dependency Check [22]: suporta projetos em Java, .NET, Ruby, Node.js, Python e C/C++ (que utilizam CMake ou autoconf). Possui como fonte de dados o NVD e permite integração com Maven, Ant e Jenkins.

OWASP Wordpress Vulnerability Scanner [23]**:** verifica sites em Wordpress em busca de vulnerabilidades nas configurações, versão do Wordpress, plugins e temas.

Palamida [24]**:** as versões Standard e Enterprise disponibilizam a verificação de vulnerabilidades em projetos que utilizam C/C++, C#, Delphi, Go, Java, Lua, PHP, Python, Ruby, dentre outras linguagens. As vulnerabilidades são obtidas através do banco de dados NVD.

Retire.js [21]**:** verifica se bibliotecas Javascript e módulos Node.js utilizados por projetos web possuem alguma vulnerabilidade conhecida. A lista de vulnerabilidades conhecidas é mantida pelo projeto e engloba informações publicadas no Node Security Project, GitHub, fóruns do Google Groups, entre outros.

SRC:CLR [29]**:** utiliza como fonte de dados de vulnerabilidades logs de sistemas de controle de versão, ferramentas de gerenciamento de defeitos, listas de e-mails e bancos de dados de vulnerabilidades. Suporta linguagens como Java, JavaScript, Python, Ruby, ferramentas de controle de versão, ferramentas de gerenciamento de dependências, e automação e integração contínua. O resultado da análise, além de sinalizar os componentes vulneráveis, indica se o projeto utiliza métodos vulneráveis do componente e sugere sua atualização.

The Victims Project [17]**:** disponibiliza um banco de dados de vulnerabilidades de componentes Java mantido pela Red Hat e plugins para Maven e Ant que identificam os componentes vulneráveis.

Veracode Software Composition Analysis [31]: analisa arquivos binários e suporta linguagens como Java e .NET. O resultado da análise, além de listar as vulnerabilidades dos componentes, apresenta sua versão mais recente. Obtém as vulnerabilidades do banco de dados NVD.

WhiteSource [33]: detecta vulnerabilidades durante o processo de construção da aplicação. Envia alertas quando são adicionados novos componentes com vulnerabilidades, e quando são disponibilizadas novas versões dos componentes. Através da ferramenta é possível verificar as dependências de cada versão dos componentes utilizados.

2.2 Requisitos

Os requisitos foram obtidos através da leitura de relatórios, observações realizadas sobre outras pesquisas e através da comparação entre ferramentas similares a esta proposta. Uma breve descrição de cada requisito funcional (RF) e não funcional (RNF) é apresentada a seguir. Tais requisitos serão utilizados no comparativo entre as ferramentas (Tabela 1).

RF01: utilizar e manter fontes de dados atualizadas. Manter as fontes de vulnerabilidades atualizada periodicamente é importante, pois uma vez que a vulnerabilidade é publicada, explorá-la se tornará mais fácil. O Relatório de Investigações de Violações de Dados publicado em 2015 pela Verizon aponta que aproximadamente 50% das vulnerabilidades foram identificadas após 1 mês da sua publicação [32].

RF02: integração de fontes de dados distintas de vulnerabilidades. Utilizar diferentes fontes de dados é importante para viabilizar informações sobre novas vulnerabilidades ou complementares sobre determinada vulnerabilidade. Atualmente, as fontes de dados de vulnerabilidades são heterogêneas e descentralizadas. Por exemplo, algumas vulnerabilidades demoram a ser disponibilizadas no *National Vulnerability Database* (NVD) por não possuírem identificador [19].

RF03: permitir extração de dependências de diferentes tipos de projetos. Permitir a extração e análise de dependências de diferentes tipos de projetos em diferentes linguagens de programação é um critério ideal para expandir o escopo de análise da ferramenta [24] [29].

RF04: listar novas versões do componente. Disponibilizar novas versões do componente que corrigem a vulnerabilidade é uma característica fundamental, pois expande e potencializa a resolução de vulnerabilidades do projeto [6].

RF05: listar incompatibilidades entre a versão atual e a nova versão do componente. Em [4], Candiru sugeriu a implementação deste critério como trabalho futuro. Ele argumenta que sinalizar incompatibilidades entre versões do componente auxilia na identificação de possíveis alterações a serem realizadas no código-fonte ao alterar a versão do componente.

RF06: identificar propagação de atualizações. Em [4], Candiru propões indicar a propagação de atualizações no projeto. Ele sustenta que esta identificação potencializa um melhor planejamento da atualização pois a atualização de um componente pode implicar na atualização de suas dependências

RF07: listar vulnerabilidades das outras versões do componente. É importante identificar as vulnerabilidades das novas versões do componente, pois, de acordo com Candiru [7],

Tabela 1: Comparativo entre ferramentas.

Ferramentas	Requisitos / Critérios Compativos							
	RF1	RF2	RF3	RF4	RF4	RF6	RF7	RNF1
dep|ct	+	+	+	+	+	+	+	+
Application Health Check	+	+	+	-	+	-	-	+
bitHound	+	-	+	+	+	+	-	-
Black Duck Hub	+	+	+	+	+	-	-	+
bundler-audit	+	+	-	+	+	-	-	-
Codenomicon AppCheck	+	+	+	+	-	-	-	-
Contrast	+	?	+	NA	+	-	-	-
Gemnasium	+	+	+	+	+	-	-	-
Nexus Lifecycle	+	+	+	+	+	-	-	+
Node Security Project	+	+	-	+	-	-	-	-
OWASP D. C.	+	-	+	+	-	-	-	-
OWASP W. V. S.	+	-	-	NA	-	-	-	-
Palamida	+	-	+	-	-	-	-	-
Retire.js	+	+	-	+	-	-	-	-
SRC:CLR	+	+	+	+	+	-	-	+
The Victims Project	+	-	+	+	-	-	-	-
Veracode Soft. Comp. Analysis	+	-	+	+	+	-	-	-
WhiteSource	+	?	+	+	-	-	+	+

Legenda: + Suportado; - Não Suportado; ? Não Identificado; NA Não Aplicável

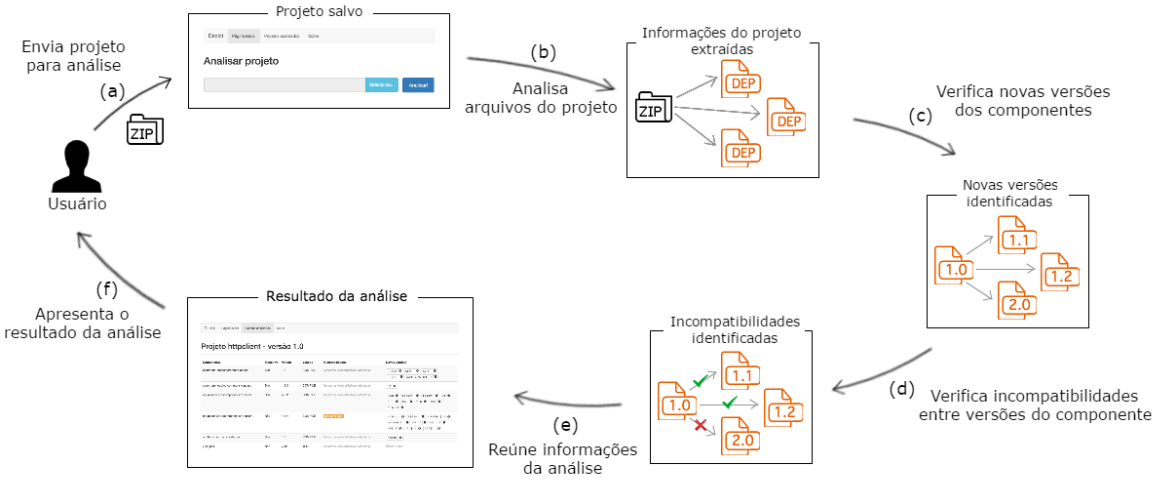

Figura 1. Fluxo geral da análise de um projeto.

versões mais recentes de componentes podem conter as mesmas vulnerabilidades das versões anteriores, ou novas vulnerabilidades mais críticas.

RNF1: possibilidade de integrar a ferramenta no processo de desenvolvimento de software. Possibilitar a integração da ferramenta no processo de desenvolvimento de software é importante para viabilizar a utilização com as ferramentas que os desenvolvedores já utilizam [13]. Em [35], Wurster e Van Oorschot reportam que as ferramentas são executadas geralmente de forma independente e, desse modo, não há como garantir que todos os desenvolvedores utilizem apenas uma ferramenta de forma unificada e eficaz.

2.3 Análise Comparativa

A Tabela 1 apresenta a uma análise comparativa entre as ferramentas descritas na Seção 2.1 e a ferramenta proposta descrita na Seção 3, utilizando os critérios apresentados na Seção 2.2. As ferramentas *Contrast* e *OWASP Wordpress Vulnerability Scanner* foram classificadas como não aplicáveis para o critério RNF01, integrar a ferramenta no processo de desenvolvimento, porque realizam a análise do projeto em tempo de execução. A ferramenta proposta possui maior flexibilidade, permitindo que novas integrações com fontes de dados sejam adicionadas (RF02). Isto viabiliza que o usuário selecione as fontes que deseja utilizar com informações específicas, tais como as bibliotecas e frameworks implementados e utilizados pela organização. Nota-se também que informações sobre as vulnerabilidades das outras versões do componente (RF04), propagação de atualizações e incompatibilidades entre versões (RF05), são disponibilizadas por poucas ferramentas. Por fim, todas as características relatadas na Seção anterior são contempladas na ferramenta dep|ct, e auxiliam na visualização de informações relevantes a serem levadas em consideração na atualização dos componentes. A seguir, a ferramenta dep|ct é apresentada.

3 FERRAMENTA PROPOSTA

Esta Seção tem como objetivo apresentar a dep|ct, uma ferramenta para identificar componentes vulneráveis conhecidos utilizados pelas aplicações e auxiliar na decisão sobre a atualização de tais componentes. Para isto, a Seção 4.1 apresenta uma visão geral da ferramenta. A Seção 4.2 apresenta as etapas realizadas para o projeto da ferramenta. A Seção 4.3 apresenta os aspectos de implementação.

3.1 Visão Geral da Ferramenta

A ferramenta proposta lê o código-fonte de projetos de software e identifica os componentes configurados como dependências, suas vulnerabilidades e as incompatibilidades entre as versões atual e mais recentes. Após isso, a ferramenta recomenda versões de componentes que sejam mais recentes, não apresentam – ou apresentam menos – vulnerabilidades conhecidas e causem menor impacto de atualização considerando as incompatibilidades entre versões. A Fig. 1 apresenta uma visão geral da ferramenta com os principais passos executados para realizar a análise de um projeto, descritos a seguir:

a) O resultado da análise é apresentado ao usuário;
b) O usuário submete um projeto para análise;
c) O sistema extrai informações sobre o projeto, como nome, versão e componentes configurados como dependências;
d) O sistema verifica se os componentes possuem versões mais recentes;
e) O sistema verifica se há incompatibilidades entre a versão do componente utilizada pelo projeto e versões mais recentes;
f) O sistema reúne as informações geradas pela análise, verifica se os componentes possuem vulnerabilidades conhecidas e recomenda uma configuração de versões dos componentes;

Os passos b, c e d podem ser customizados através da implementação de interfaces, que caracterizam pontos de extensão do comportamento da ferramenta.

3.2 Arquitetura e Componentes

Visando um projeto flexível, extensível e com alto grau de reuso, propõe-se uma arquitetura baseada em componentes organizada em camadas. A arquitetura possui dois módulos principais – *core* e *web* – cuja disposição em camadas e componentes em alto nível são apresentados na Fig. 2.

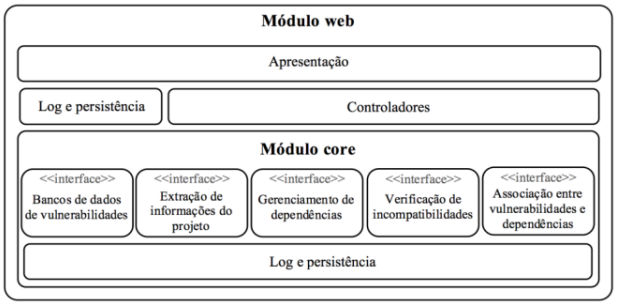

Figure 2: Modulo da web e seus componentes em anto nível.

A Figura 3, por sua vez, apresenta o componente principal da ferramenta com suas interfaces requeridas e providas. O módulo *core* é responsável por coordenar o fluxo de execução da aplicação. Este módulo não possui interface gráfica e tem como finalidade a utilização por outros módulos ou aplicações. O principal componente do módulo *core* é a *Engine*, o qual é responsável pela análise dos projetos e sincronização das fontes de dados de vulnerabilidades.

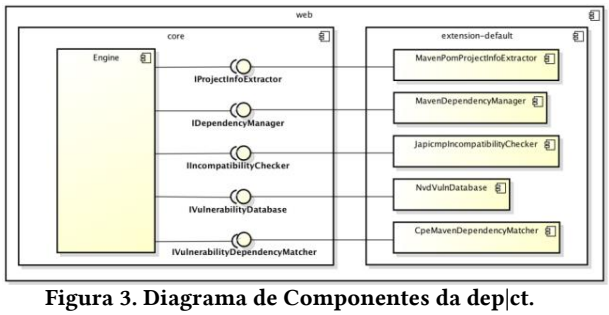

Figura 3. Diagrama de Componentes da dep|ct.

A *Engine* exige a implementação de determinadas interfaces, que se caracterizam como pontos de extensão onde o comportamento da ferramenta poderá ser alterado. Estas interfaces são descritas abaixo:

a) ***IProjectInfoExtractor***: define as operações que devem ser implementadas para extração de informações (como nome e versão) e dependências do projeto a ser analisado;
b) ***IDependencyManager***: define as operações que devem ser implementadas para verificar as versões disponíveis para determinado componente;
c) ***IIncompatibilityChecker***: define as operações a serem implementadas por componentes que verificam incompatibilidades entre duas versões de um mesmo componente;
d) ***IVulnerabilityDatabase***: define as operações que devem ser implementadas para obtenção das vulnerabilidades a partir de uma fonte externa. O componente que implementa esta interface também poderá sinalizar os componentes e versões afetados pela vulnerabilidade; e
e) ***IVulnerabilityDependencyMatcher***: define as operações que devem ser implementadas para realizar a associação entre vulnerabilidades e dependências. Esta interface deve ser implementada quando a integração com as fontes de dados de vulnerabilidades não realizar esta associação.

O módulo *web* utiliza o módulo *core* e é responsável por disponibilizar a interface Web (camada de apresentação) para o usuário final. A comunicação entre a camada de apresentação e a aplicação é realizada através de serviços REST, disponibilizados pela camada de controladores. O módulo web utiliza também o módulo secundário *extension-default*, que fornece componentes com implementações padrão.

3.3 Aspectos de Implementação

A ferramenta foi implementada na linguagem Java 0 e utiliza Maven 0 para o gerenciamento da construção e dependências do projeto. Foi utilizado o framework Spring, incluindo Spring Boot para simplificar a configuração e tornar a aplicação *stand-alone*, Spring Web para implementação da API REST e Spring Data JPA em conjunto com o framework *Hibernate* para a camada de persistência. A interface web foi desenvolvida utilizando os frameworks AngularJS e Bootstrap. Para o armazenamento dos dados foi utilizado o banco de dados relacional PostgreSQL. Tais tecnologias foram escolhidas porque são amplamente utilizadas na indústria para o desenvolvimento de aplicações corporativas, incluindo sistemas de informação.

A Figura 4 apresenta a fórmula utilizada para calcular o custo de uma configuração. Na fórmula são consideradas as vulnerabilidades de todas as dependências (transitivas e não transitivas) com escopo diferente de "teste" e as incompatibilidades apenas das dependências não transitivas.

$$CustoVulnerabilidades = \sum_{i=0}^{numVulns} (pontuacaoVuln_i) + numVulns * 5$$

$$CustoIncompatibilidades = numIncompatibilidades$$

$$CustoConfiguracao = CustoVulnerabilidades + CustoIncompatibilidades$$

Figura 4. Cálculo do custo de uma configuração.

4 AVALIAÇÃO DA FERRAMENTA

Esta Seção apresenta uma avaliação empírica da ferramenta. A Seção 5.1 apresenta o método utilizado para a avaliação. A Seção 5.2 apresenta os resultados, incluindo melhorias identificadas para a ferramenta. A Seção 5.3 apresenta uma análise crítica da ferramenta baseada nos resultados obtidos.

4.1 Método de Avaliação

O propósito da avaliação da ferramenta é apresentar resultados obtidos a partir da análise dos projetos utilizando a implementação padrão dos pontos de extensão da ferramenta, demonstrando que a ferramenta auxilia na decisão sobre a atualização de componentes vulneráveis conhecidos.

Para avaliar a ferramenta foram selecionados 2 projetos: *Apache Storm* e *Activiti*. Estes projetos foram escolhidos pois tratam-se de aplicações robustas, desenvolvidas em Java, que utilizam Maven para o gerenciamento de dependências, estão publicadas no GitHub e possuem propósitos distintos. Como estes projetos são compostos por vários módulos (vários arquivos POM do Maven) e a implementação padrão permite analisar apenas um arquivo POM de cada vez, foi selecionado um módulo de cada projeto para ser analisado.

Foram selecionadas 3 versões de cada projeto da seguinte forma: primeira versão na qual o módulo foi introduzido, versão intermediária – considerando a primeira e a última versão selecionadas – e última versão. A Tabela 2 apresenta uma breve descrição sobre cada projeto, além dos módulos e versões selecionadas, incluindo a data da última alteração da versão.

Tabela 2. Descrição dos projetos selecionados.

Característica	Apache Storm	Activiti
Descrição	Sistema de computação distribuído que permite processar dados em tempo real.	Plataforma de gerenciamento de processos de negócio (BPM) e fluxo de trabalho.
Módulo	Storm-core	Activiti-engine
Website	http://storm.apache.org/	http://activiti.org/
Repositório	https://github.com/apache/storm	https://github.com/Activiti/Activiti
Versões Selecionadas	0.9.3 (19/11/2014) 0.10.0 (23/10/2015) 1.1.0 (15/07/2016)	5.0 (31/01/2011) 5.14 (21/10/2013) 5.21.0 (13/06/2016)

Para submeter os projetos foi obtido o arquivo compactado contendo todos os arquivos do projeto na versão selecionada através da interface do GitHub. Após isso, o arquivo foi descompactado e foram removidas todas as pastas não relativas ao módulo selecionado, criando-se um novo arquivo compactado apenas com o módulo selecionado para análise e demais arquivos da raiz do projeto. A descrição das métricas extraídas são apresentadas na Tabela 3. Nenhuma das métricas considera dependências com escopo de testes.

Tabela 3. Descrição das Métricas analisadas.

Métrica	Descrição
#Deps	Número total de dependências do projeto
#DepsTransitivas	Número de dependências transitivas do projeto
#DepsComVulnerabilidades	Número total de dependências com vulnerabilidades
#DepsTransitivasComVulnerabilidades	Número de dependências transitivas com vulnerabilidades
#Vulns	Número total de vulnerabilidades detectadas
#VulnsTransitivas	Número de vulnerabilidades detectadas introduzidas através de dependências transitivas
#VulnsOWASPDependencyCheck	Número total de vulnerabilidades detectadas pela ferramenta *OWASP Dependency Check*

4.2 Método de Avaliação

A Tabela 4 apresenta os valores obtidos de cada métrica para cada versão dos projetos analisados. Para critérios comparativos, o número de vulnerabilidades encontradas pela ferramenta OWASP *Dependency Check* também foi adicionado. De acordo com os dados, pode-se notar que há crescimento no número de dependências para os dois módulos e que há grande divergência entre o número de vulnerabilidades encontradas pela ferramenta proposta e pela OWASP *Dependency Check*, ocasionada pelas duas limitações descritas a seguir.

Tabela 4. Métricas obtidas através da análise dos projetos

Módulo Apache Storm		
Versão	**1.1.0**	**Recomendação**
#VuIns	6	3
#VuInsTransitivas	1	1
#DepsComVulnerabilidades	4	2
#DepsTransitivasComVulnerabilidades	1	1
Módulo Activiti Activiti-Engine		
Versão	**1.1.0**	**Recomendação**
#VuIns	3	2
#VuInsTransitivas	2	1
#DepsComVulnerabilidades	3	2
#DepsTransitivasComVulnerabilidades	2	1

A partir da execução das análises verificou-se que as implementações padrão não identificaram vulnerabilidades que não possuem CPEs listados para todas as versões afetadas, como é o caso do CVE-2015-5262, que afeta todas as versões do componente Apache *HttpComponents HttpClient* anteriores à versão 4.3.6. Isto ocorre devido ao formato de representação dos dados nos arquivos do NVD, pois a ferramenta utiliza os *feeds* disponibilizados na versão 2.0 e este formato não possui nenhuma indicação quando a vulnerabilidade afeta versões anteriores a do CPE indicado. Como melhoria, o componente responsável pela integração com o NVD poderia extrair esta

informação do *feed* na versão 1.2.1 ou da descrição da vulnerabilidade.

Outra limitação identificada diz respeito à sinalização de versões incompletas no CPE da vulnerabilidade, como é o caso do CVE-2007-5613, que indica as versões compostas por 1 e 2 dígitos *cpe:/a:mortbay_jetty:jetty:6* e *cpe:/a:mortbay_jetty:jetty:6.1*. Como a ferramenta adiciona caracteres curinga no final da versão indicada no CPE para obter componentes com complementos de versão como "*release*" e "final", outras versões deste componente também são incluídas na lista de componentes vulneráveis (por exemplo, 6.1.26).

Tabela 5. Configuração recomendada nos projetos.

Módulo Apache Storm			
Versão	**0.9.3**	**0.10.0**	**1.1.0**
#Deps	63	66	75
#DepsTransitivas	38	26	28
#DepsComVulnerabilidades	5	4	4
#DepsTransitivasCom Vulnerabilidades	3	1	1
#VuIns	8	6	6
#VuInsTransitivas	5	1	1
#VuInOWASPDependencyPack	15	16	15
Módulo Activiti			
Versão	**5.0**	**5.14**	**5.21.0**
#Deps	21	34	38
#DepsTransitivas	12	19	20
#DepsComVulnerabilidades	4	3	3
#DepsTransitivasCom Vulnerabilidades	2	2	2
#VuIns	9	9	3
#VuInsTransitivas	7	8	2
#VuInOWASPDependencyPack	8	14	5

Estes dois aprimoramentos da implementação padrão são importantes para diminuir o número de falsos positivos e falsos negativos gerados pela ferramenta. Para verificar se os dados apresentados pela ferramenta – e utilizados na recomendação de configuração – auxiliam a decidir sobre a atualização dos componentes com o objetivo de melhorar o quadro de vulnerabilidades das aplicações, a Tabela 5 apresenta as métricas de vulnerabilidades das configurações recomendadas considerando a última versão de cada módulo analisado.

As vulnerabilidades restantes do módulo *storm-core* são relativas ao componente *Apache ZooKeeper* (que não possui novas versões) e ao componente Apache *HttpComponents HttpClient*, que possui versão mais recente, porém, como é uma dependência transitiva do componente *Apache Thrift* e este não possui novas versões, não foi considerada na recomendação.

A vulnerabilidade removida do módulo *activiti-engine* foi consequência da atualização do componente *spring-beans* da versão 4.1.5.RELEASE para a versão 4.3.3.RELEASE. As duas vulnerabilidades restantes são falsos-positivos introduzidos pela forma de associação dos componentes afetados pela vulnerabilidade.

4.3 Análise Crítica da Ferramenta

De modo geral, considera-se que a ferramenta apresentada pode ser utilizada para auxiliar na decisão sobre a atualização de componentes vulneráveis conhecidos. Porém, recomenda-se melhorias para que a mesma possa ser amplamente utilizada na indústria relacionadas a usabilidade e a efetividade da ferramenta. Além das melhorias citadas anteriormente, outras melhorias foram identificadas através da submissão dos projetos e análise dos resultados:

a) Considerar o impacto da vulnerabilidade no contexto de uso do componente na aplicação, tanto para sinalizar a vulnerabilidade, quanto para recomendar uma nova configuração;

b) Integração com sistemas de controle de versão e notificação por e-mail dos resultados da análise e inclusão de novas vulnerabilidades, evitando assim a intervenção manual para execução da ferramenta;

c) Apresentar resumo com métricas do projeto analisado, incluindo número de dependências e vulnerabilidades, tanto da configuração atual quanto da configuração recomendada;

d) Exibir a árvore de dependências, pois atualmente não é possível identificar a origem da inclusão de uma dependência no projeto;

e) Aprimorar o processamento dos projetos, pois quanto maior o número de dependências e versões destas dependências, maior o tempo de processamento necessário para analisar o projeto.

5 CONSIDERAÇÕES FINAIS

A utilização de componentes vulneráveis conhecidos é um problema que ganhou maior destaque após ser incluído no relatório da OWASP em 2013. Várias ferramentas auxiliam a identificar a utilização de tais componentes, porém poucas apresentam informações que auxiliam na mitigação das vulnerabilidades através da atualização do componente.

Este artigo apresentou a *dep|ct*, uma ferramenta flexível que identifica a utilização de componentes vulneráveis conhecidos, apresenta informações que auxiliam na decisão sobre a atualização de tais componentes e recomenda uma configuração de versões com o objetivo de mitigar as vulnerabilidades adicionadas aos projetos através das dependências vulneráveis. Foram descritas as implementações padrão das interfaces requeridas pelo módulo core, que utilizam o NVD como fonte de dados de vulnerabilidades e permitem a análise de projetos Java que utilizam Maven para o gerenciamento de dependências.

A avaliação empírica realizada sobre a implementação padrão demonstra que a ferramenta pode ser utilizada para auxiliar na decisão sobre a atualização dos componentes vulneráveis conhecidos identificados. As limitações encontradas na avaliação são relativas à implementação padrão e não à arquitetura da ferramenta e são consideradas melhorias a serem implementadas. Tanto a implementação padrão quanto a ferramenta ainda necessitam de melhorias, sendo as principais: aprimorar a associação entre vulnerabilidades e dependências, considerar o impacto da vulnerabilidade no contexto de uso do componente na aplicação, permitir integração com sistemas de controle de versão e enviar notificações por e-mail.

AGRADECIMENTOS

Gostaríamos de agradecer a Unisinos por fornecer espaço e os recursos para executar essa pesquisa.

REFERENCES

[1] Allen, Julia. Why is Security a Software Issue?. EDPACS: The EDP Audit, Control, and Security Newsletter, Pittsburgh, v. 36, n. 1, (Aug. 2007) p. 1-13. Disponível em: <http://www.tandfonline.com/doi/ref/10.1080/07366980701500734>. Acesso em: 13 fev. 2018.

[2] Alqahtani, S. S., Eghan, E. E., & Rilling, J. Tracing known security vulnerabilities in software repositories–A Semantic Web enabled modeling approach. *Science of Computer Programming*, *121*, 2016, 153-175.

[3] Bithound Inc. Features. Kitchener, 2016. Disponível em: <https://www.bithound.io/features>. Acesso em: 19 fev. 2018.

[4] Black Duck | Hub. Burlington, 2016. Disponível em: <https://info.blackducksoftware.com/rs/872-OLS-526/images/BlackDuck_HUB_UL.pdf>. Acesso em: 19 fev. 2018.

[5] Bundler-Audit. Patch-level verification for Bundler, 2016. Disponível em: <https://github.com/rubysec/bundler-audit>. Acesso em: 19 fev. 2018.

[6] Cadariu, M. Tracking Known Security Vulnerabilities in Third-party Components, Netherlands. 2014. 86 f. M. Sc. Dissertation - Programa de Pós-Graduação em Ciência da Computação, Delft University of Technology, Holanda, 2014.

[7] Cadariu, M.; Bowers, E.; Visser, J.; Deuresen, A. V. Tracking Known Security Vulnerabilities in Proprietary Software Systems. In: 2015 *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering*, (Montreal, QC, Março 2015), SANER, 516-519.

[8] Cheikes, B. A.; Waltermire, D.; Scarfone, K. Common Platform Enumeration: Naming Specification Version 2.3. National Institute of Standards and Technology (NIST), Gaithersburg, Aug. 2011.

[9] Codenomicon LTD. AppCheck - Frequently Asked Questions. Oulu. Disponível em: <http://www.codeno micon.com/products/appcheck/faq/>. Acesso em: 19 fev. 2018.

[10] Contrast Security. Complete Coverage of Today's Modern Applications. Palo Alto, 2016. Disponível em: <https://www.contrast security.com/supported-technologies>. Acesso em: 19 fev. 2018.

[11] Durumeric, Zakir et al. The Matter of Heartbleed. *Proceedings of the 2014 Conference on Internet Measurement Conference*, (Vancouver, Quebec, Nov. 2014), 475-488.

[12] Financial Services Information Sharing and Analysis Center. Appropriate Software Security Control Types for Third Party Service and Product Providers., Oct. 2015.

[13] Fonseca, V. S.; Barcellos; M. P; Almeida Falbo, R. Tools Integration for Supporting Software Measurement: A Systematic Literature Review. *iSys-Revista Brasileira de Sistemas de Informação*, *8*,4, 2016, 80-108.

[14] Gemnasium. Features. Paris, 2016. Disponível em: <https://gemnasium.com/features>. Acesso em: 10 jan. 2018.

[15] Gosling, J.; Joy, B.; Steele, G.; Bracha, G.; Buckley, A. The Java Language Specification: Java SE 8 Edition. Redwood City: Oracle America, Inc. and/or its affiliates, 2015. Disponível em: <http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf>. Acesso em: 10 jan. 2018.

[16] Mcgraw, G. Software Security: Building Security In. Upper Saddle River, NJ: Addison-Wesley, 2006.

[17] Milner, Steve; Victims Project Team. Why Does This Exist?. 2016. Disponível em: <https://victi.ms/about.html>. Acesso em: 19 fev. 2018.

[18] Murtaza, S. S.; Khreich, W.; Hamou-Lhadj, A.; Bener, A. B. Mining trends and patterns of software vulnerabilities. *Journal of Systems and Software*, *117*, 2016, 218-228.

[19] National Vulnerability Satabase (NVD). NVD Frequently Asked Questions. Disponível em: <https://nvd.nist.gov/faq>. Acesso em: 9 fev. 2018.

[20] Node Security. Tools. Richland. Disponível em: <https://nodesecurity.io/tools>. Acesso em: 19 fev. 2018.

[21] Oftedal, E. Retire.js: What you require you must also retire. 2016. Disponível em: <http://retirejs.github.io/r etire.js/>. Acesso em: 10 jan. 2018.

[22] Open Web Application Security Project (OWASP). OWASP Dependency Check. Bel Air, June 16, 2016. Disponível em: <https://www.owasp.org/index.php/OWASP_Dependency_Check>. Acesso em: 10 jan. 2018.

[23] Open Web Application Security Project (OWASP). OWASP Wordpress Vulnerability Scanner Project. Bel Air, Dec. 8, 2015. Disponível em: <https://www.owasp.org/index.p hp/OWASP_Wordpress_Vulnerability_Scanner_Project>. Acesso em: 10 jan. 2018.

[24] Palamida, Inc. Standard Edition. San Francisco, 2015. Disponível em: <http://www.palamida.com/files /Palamida-Standard-Edition-Datasheet.pdf>. Acesso em: 10 jan. 2018.

[25] PCI Security Standards Council. Payment Card Industry (PCI) Data Security Standard. Apr. 2015. Disponível em: <https://www.pcisecuritystandards.org/documents/PCI_DSS_v3-1.pdf>. Acesso em: 10 jan. 2018.

[26] Plate, H.; Ponta, S. E.; Sabetta, A. Impact Assessment for Vulnerabilities in Open-source Software Libraries. In: *2015 IEEE International Conference on Software Maintenance and Evolution.* (Bremen, Sept. 2015), (ICSME), 411-420.

[27] Sonatype Inc. Application Health Check. Fulton, 2008. Disponível em: <http://www.sonatype.com/download-application-health-check>. Acesso em: 10 fev. 2018.

[28] Sonatype Inc. Nexus IQ Server Documentation. Fulton, June 16, 2016. Disponível em: <http://books.sonatype. com/sonatype-clm-book/pdf/book-clm.pdf>. Acesso em: 19 fev. 2018.

[29] Sourceclear. Better Science: Better science on over 5 million libraries. San Francisco, 2016. Disponível em: <https://srcclr.com/features/science>. Acesso em: 10 jan. 2018.

[30] The Apache Software Foundation (ASF). Maven Getting Started Guide. 2016. Disponível em: <https://maven.apache. org/guides/getting-started/index.html>. Acesso em: 10 jan. 2018.

[31] Veracode. Software Composition Analysis. Burlington. Disponível em: <https://info.veracode.com/data sheet-software-composition-analysis.html>. Acesso em: 10 jan. 2018.

[32] Verizon Enterprise Solutions. 2015 Data Breach Investigations Report, Apr. 2015. Disponível em: <http://news.verizonenterprise.com/2015/04/2015-data-breach-report-info/>. Acesso em: 11 fev. 2018.

[33] Whitesource Software. Continuously Audit Open Source Components in Your Code. New York, 2016. Disponível em: <http://www.whitesourcesoftware.com /open_source_scann ing_software/>. Acesso em: 20 fev. 2018.

[34] Williams, J.; Wichers, D. OWASP Top-10 2013. [S.l.], Feb. 2013. Disponível em: <https://storage.googleapis.com/google -code-archive-downloads/v2/code.google.com/owasptop 10/OWASP%20Top%2010%20-%202013.pdf>. Acesso em: 1 fev. 2018.

[35] Wurster, G.; Van Oorschot, P. C. The Developer is the Enemy. In: *Proceedings of the 2008 Workshop on New Security Paradigms.* (Lake Tahoe, California, Sept. 2008) ACM, New York, 89-97, Sept. 2008.