Testes de Segurança em Aplicações *Web Open Source*: Um Estudo Comparativo entre Ferramentas de *Issue Tracking*

Ramon Oliveira Silva
Universidade Federal de Viçosa
Florestal, MG
ramon.o.silva@ufv.br

Gláucia Braga e Silva
Universidade Federal de Viçosa
Florestal, MG
glaucia@ufv.br

ABSTRACT

This study aims to evaluate security vulnerabilities in web open source tools for issue tracking, based on the criteria proposed by ISO/IEC 27000 series standards and OWASP Community. The Owasp Guide was used to conduct the tests considering 10 most frequent vulnerabilities and the Owasp Zap tool was used to automate the tests. The tests were applied on three open source issue tracking tools namely Redmine, Mantis Bug Tracker and Bugzilla. The tests reports revealed few security vulnerabilities in the Redmine tool, but a lot of them in the context of Mantis Bug Tracker and Bugzilla.

CCS CONCEPTS

• **Security and privacy → Authentication**; **Access control**; **Web application security**; **Cryptography**;

KEYWORDS

Vulnerabilidades de Segurança, Owasp, Ferramentas de Teste

ACM Reference Format:
Ramon Oliveira Silva and Gláucia Braga e Silva. 2019. Testes de Segurança em Aplicações *Web Open Source*: Um Estudo Comparativo entre Ferramentas de *Issue Tracking*. In *Proceedings of ACM SAST conference (SAST).* ACM, New York, NY, USA, Article 4, 8 pages. https://doi.org/10.475/123_4

1 INTRODUÇÃO

Os sistemas *web* tiveram um crescimento exponencial nos últimos anos, e devido à facilidade de acesso, por ser um espaço aberto, são bastante vulneráveis e estão sujeitos à invasões, já que existem milhares de pessoas mal-intencionadas que tentam invadir estes sistemas todos os dias.

Existe uma fonte de fragilidades chamada CVE (*Common Vulnerabilities and Exposures*)[1], que é um projeto *open source* e mantém uma base de dados com todas as vulnerabilidades que são encontradas ao longo dos anos em várias ferramentas.

Dentre as ferramentas *web* que merecem atenção quanto aos aspectos de segurança, destacam-se as ferramentas de *issue tracking*. Tais ferramentas são responsáveis por registrar bugs de aplicações diversas, constituindo assim um importante repositório de fragilidades, incluindo segurança, que devem ser devidamente corrigidas. Para exemplificar, em outrubro de 2014, a Mozilla corrigiu uma falha de segurança crítica no Bugzilla[2]. O problema permitia que a aplicação fosse invadida por hackers, que teriam acesso ao banco de dados e poderiam descobrir falhas na aplicação[8]. Em 2015, também foi descoberta uma vulnerabilidade na ferramenta Mantis Bug Tracker (MantisBT)[3], que permitia a inserção de scripts maliciosos (*Cross site scripting*) em um campo do sistema[5]. Já em 2017, na ferramenta Redmine[4], uma falha permitia a injeção de códigos arbitrários envolvendo a renderização de mensagens em Flash[6].

Sendo assim, torna-se fundamental a realização de testes de segurança em ferramentas de *issue tracking*, para que suas fragilidades sejam identificadas, corrigidas e seu uso não comprometa os projetos controlados nas mesmas.

O objetivo deste trabalho é avaliar vulnerabilidades nas ferramentas *web open source* de *issue tracking* Bugzilla, MantisBT e Redmine, com o intuito de verificar se ainda existem vulnerabilidades de segurança nas mesmas e estabelecer mais um critério de escolha que possa apoiar gerentes de projetos no momento da adoção de tais ferramentas. Para isso, serão realizados testes de segurança automatizados com base nos critérios definidos pelas normas da série ISO/IEC 27000 e pela comunidade Owasp[5].

Os testes em ferramentas de *issue tracking* são relevantes, pois essas ferramentas servem para gerenciar *bugs* que são encontrados pelos testadores. Esses *bugs* não podem ser acessados por um usuário qualquer, já que o mesmo pode utilizar essas vulnerabilidades para atacar a própria ferramenta. Geralmente, esses *bugs* são corrigidos e lançados em novas versões, o que possibilita ao hacker explorar essa vulnerabilidade na versão reportada. Além disso, esta pesquisa se mostra relevante pelo motivo de não terem sido encontrados trabalhos na literatura abordando a realização de testes de segurança em ferramentas de *issue tracking*, apesar de serem amplamente adotadas no contexto de um projeto de software e de já terem sido encontradas algumas vulnerabilidades de segurança nas mesmas.

O texto do trabalho está organizado da seguinte forma: na seção 2, são apresentadas as vulnerabilidades de segurança em aplicações *web*. Na seção 3, são discutidas as ferramentas de apoio aos testes de segurança. Na seção 4, são discutidos os trabalhos relacionados. Na seção 5, é apresentada a base conceitual para a compreensão dos testes de segurança em aplicações *web*, os resultados, análises e discussões. Por fim, a seção 6 traz as conclusões e trabalhos futuros.

[1] https://www.cvedetails.com/

[2] https://www.bugzilla.org/
[3] https://www.mantisbt.org/
[4] https://www.redmine.org/
[5] https://www.owasp.org/

SAST, 2019, Bahia

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

2 VULNERABILIDADES DE SEGURANÇA DE APLICAÇÕES WEB

Vulnerabilidades são formas pelas quais as ameaças se manifestam em um sistema. O processo de avaliação de vulnerabilidades consiste em identificar, avaliar e apresentar formas de reduzir tais ameaças, melhorando a capacidade do sistema para lidar com possíveis falhas futuras[10].

As aplicações *web* requerem um alto grau de segurança. Logo, surge a necessidade de uma informação segura e imparcial. Nesse cenário temos a OWASP (*Open Web Application Security Project*), que é uma comunidade aberta dedicada a encontrar e combater problemas de vulnerabilidades em sistemas *web*. Ela cria e disponibiliza de forma gratuita artigos, metodologias, documentos, ferramentas e tecnologias no campo da segurança de aplicações *web*[13].

A comunidade OWASP publica o TOP 10[14] com o intuito de educar desenvolvedores, designers, arquitetos e organizações a respeito das consequências das vulnerabilidades mais comuns encontradas em aplicações *web*. Para isso, ela disponibiliza um documento de alto-nível que foca nas dez vulnerabilidades mais críticas de aplicações *web*, mais frequentes a cada ano. Dentre essas, as selecionadas neste trabalho para realização dos testes de segurança são.

2.1 Cross Site Scripting (XSS)

A vulnerabilidade *Cross Site Scripting* ocorre sempre que uma aplicação obtém as informações fornecidas pelo usuário e as envia de volta ao navegador sem realizar validação ou codificação daquele conteúdo. O XSS permite aos atacantes executarem scripts no navegador da vítima, o qual podem roubar sessões de usuário, desconfigurar sites, redirecionar a vítima para sites não confiáveis e até controlar o navegador do usuário. O script malicioso normalmente é escrito em Java Script, mas qualquer linguagem de script suportada pelo navegador também pode ser utilizada. O XSS é classificado em *refletido*, *armazenado* e *baseado em DOM* [14]:

- O *refletido* é o tipo mais frequente de ataque XSS, sendo iniciado com a ação de clicar em um link que possui parâmetros modificados e que levam a ações não desejadas. Os passos básicos para executar esse ataque é modelar um URL malicioso, identificar os campos de entrada de dados e posteriormente convencer a vítima a acessá-lo[11].
- O *armazenado* ocorre quando o código malicioso fica armazenado na aplicação e é executado pelo navegador de qualquer vítima que acesse a página com tal código. O teste consiste em identificar os campos onde a entrada fornecida pelo usuário é armazenada e posteriormente exibida, como tag's de formulários.
- Já no *baseado em DOM* (*Document Object Model*), o código Java Script do site e as variáveis são manipulados ao invés dos elementos HTML. Ele permite a modificação de propriedades desses objetos diretamente no navegador da vítima, não dependendo de nenhuma interação por parte do servidor que hospeda o serviço[10].

2.2 SQL Injection

Esta vulnerabilidade acontece quando os dados fornecidos pelo usuário são enviados como parte de um comando ou consulta. Os atacantes confundem o interpretador para que o mesmo execute comandos manipulados enviando dados modificados. Caso uma entrada de um usuário seja fornecida a um interpretador sem validação ou codificação, a aplicação é vulnerável. Um ataque com sucesso pode retornar aos atacantes diversas informações sensíveis do banco de dados, além de poder permitir a manipulação dos dados, operações como inserir, editar e deletar[11].

2.3 Criptografia

O objetivo é avaliar se a aplicação *web* toma medidas necessárias para evitar interceptações de dados utilizando protocolos seguros como HTTPS, se utiliza algoritmos de criptografia para manter os dados sensíveis protegidos no banco de dados.

2.4 Autenticação

É o processo de tentar verificar a identidade de um usuário na aplicação. As funções da aplicação relacionadas com autenticação são geralmente implementadas de forma incorreta, permitindo que os invasores comprometam senhas, tokens de sessão e assumam a identidade de outros usuários com ataques de força bruta, requisições diretas à página [14].

2.5 Controle de Acesso

Serve para medir o nível de acesso aos recursos do sistema, logo um usuário comum não pode realizar tarefas que cabem a um administrador de um sistema. As aplicações precisam executar as verificações de controle de acesso no servidor quando cada função é invocada. Se forem realizadas essas verificações os invasores são capazes de manipular as requisições com o propósito de acessar a aplicação sem a devida autorização.

3 FERRAMENTAS DE APOIO AOS TESTES DE SEGURANÇA

Existem ferramentas apropriadas para minimizar as quantidades de falhas em uma aplicação. São os chamados scanners de vulnerabilidade (*Vulnerability Scanning Device (VSD)*) para aplicações *web*, ferramentas automáticas, comerciais ou *open source*, que identificam e trazem recomendações para possíveis falhas de segurança, prevenindo possíveis ataques ao sistema[10]. Elas varrem o servidor e testam uma enorme quantidade de eventos com a finalidade de obter respostas inesperadas[10]. É importante ressaltar também que os resultados encontrados podem não se tratar efetivamente de um risco. Eles são considerados respostas a um determinado evento que necessita de uma investigação mais detalhada.

Neste trabalho, para a escolha da melhor ferramenta de teste, alguns critérios foram levados em consideração, como ser *open source* e cobrir a maior parte das vulnerabilidades listadas pela Owasp. Entre as ferramentas existentes que cobriam esses critérios, foram selecionadas: Skipfish, Uniscan, Vega, W3AF e a Owasp Zap.

Costa [9] realizou um estudo entre as ferramentas Skipfish, Uniscan, Vega e W3AF, e levou em consideração três funcionalidades principais, que receberam notas de 1 a 3, onde 1 significa que não atendeu minimamente os requisitos, 2 atendeu os requisitos e 3 possui requisitos acima do necessário:

- Relatório: Deve detalhar cada vulnerabilidade, além de indicar o ponto onde a mesma aconteceu.
- Usabilidade: Facilidade de utilizar a ferramenta.
- Eficácia: Quantidade de vulnerabilidades detectadas pela ferramenta, considerando a classificação da OWASP.

Com base no estudo de Costa[9], foi acrescentada e avaliada a ferramenta *Owasp Zap*, considerando-se os mesmos critérios, conforme ilustra a Tabela 1.

Tabela 1: Avaliação das Ferramentas (Adaptado de Costa [9])

Ferramentas	Relatório	Usabilidade	Eficácia
Skipfish	3	2	3
Uniscan	2	2	3
Vega	1	3	3
W3AF	1	2	3
Owasp Zap	3	3	3

Após analisar os dados, a ferramenta *Owasp Zap* foi escolhida, com base na análise dos critérios mencionados na Tabela 1, por ter sido desenvolvida pela própria comunidade Owasp e por ter sido eleita a melhor ferramenta de segurança em 2013 e 2015 [15]. Além disso, a ferramenta possibilita salvar a sessão que está sendo executada, gerar o relatório do teste em html, salvar todas as mensagens e alertas que foram gerados.

4 TRABALHOS RELACIONADOS

Esta seção apresenta alguns trabalhos relacionados que avaliam aplicações *web* segundo as vulnerabilidades listadas pela comunidade Owasp. Não foram encontrados trabalhos relacionados a testes de segurança em ferramentas de *issue tracking*, apesar de haver um grande número de vulnerabilidades encontradas nessas ferramentas, como reportado, por exemplo, no CVE [1].

No contexto de análise de vulnerabilidades em uma aplicação *web*, Costa [9] realizou testes de segurança levando em consideração as vulnerabilidades do Owasp Top 10 em portais de governos eletrônicos, buscando associar a riqueza dos estados dos portais com seu nível de segurança, no sentido de uma relação direta entre o PIB dos estados e a segurança dos portais. Inicialmente, realizou-se uma análise de alguns scanners segundo critérios de relatório, usabilidade e eficácia. Os scanners utilizados foram Skipfish, Uniscan, Vega e o W3AF, sendo o Skipfish e o Uniscan escolhidos, além de uma ferramenta extra para scanner de portas, o NMAP. Logo, os scanners foram executados em 28 portais dos estados brasileiros e avaliados, segundo os critérios da Owasp, apresentando tabelas e gráficos comparativos. O estudo conseguiu atingir resultados significativos, negando a hipótese proposta, o portal com maior número de vulnerabilidades foi o do Rio de Janeiro.

Em um outro trabalho relacionado, Lins [10] analisou três scanners de vulnerabilidades em um ambiente de teste chamado bWAPP, que possui algumas das principais vulnerabilidades categorizadas pela Owasp. Os scanners analisados foram Arachini, Nessus e o Vega. Logo, foram apresentadas tabelas comparativas com os resultados obtidos em cada scanner, o Nessus destacou-se como melhor ferramenta avaliada, mesmo sendo paga ao contrário da Arachini e o Vega que são projetos públicos.

Já Oliveira[11], fez um estudo mais detalhado de testes de segurança em aplicações *web* segundo a comunidade Owasp. As aplicações *web* testadas foram aplicações da categoria *e-comerce*, baseadas no Owasp Top 10. São apresentadas tabelas com os resultados obtidos. O objetivo neste trabalho está em analisar a comunidade Owasp segundo seu guia de vulnerabilidades e testar alguns sistemas de *e-commerce*. As aplicações escolhidas foram de *e-comerce* por possuírem campos de nome, cpf, rg, endereço, que são de natureza críticas. Os resultados apresentaram que as aplicações possuem vulnerabilidades semelhantes.

A diferença deste trabalho para Costa [9], Lins [10] e Oliveira [11], é que a ferramenta escolhida foi o *Owasp Zap* e os ambientes de testes são três ferramentas de *issue tracking*, não se limitando a uma análise mais detalhada entre scanners.

5 TESTES DE SEGURANÇA REALIZADOS

Com base nas vulnerabilidades de segurança definidas de acordo com critérios de maiores ocorrências nos últimos anos (Seção 2), os testes analisaram as seguintes vulnerabilidades:

- Cross Site Scripting (XSS)
- SQL Injection
- Controle de Acesso
- Autenticação
- Criptografia

5.1 As ferramentas sob teste

As ferramentas avaliadas foram escolhidas segundo os critérios de popularidade, ser *open source* e possuir versão *web*. Para o critério de popularidade, foram selecionadas as ferramentas Bugzilla[6] , Mantis Bug Tracker[7] e Redmine[8], pois a cada ano elas estão sempre bem rankeadas nas dez primeiras posições das ferramentas de *issue tracking* mais populares, como citados no sites [12] e [7]. Além disso, as três ferramentas também são *open source* e possuem versão *web*.

O MantisBT é uma ferramenta *web open source* para gerenciamento de bugs, criada no ano de 2000 por Kenzaburo Ito que a desenvolveu em PHP. A ferramenta funciona em diversos bancos de dados entre eles o MySQL e o PostgreSQL, além de estar disponível para diversas plataformas como Windows, Mac e várias distribuições do Linux e possuir uma versão Mobile. Ao longo do tempo amadureceu e ganhou bastante popularidade, sendo umas das ferramentas mais populares atualmente.

O Redmine também é uma ferramenta *web open source* e permite o gerenciamento de bugs e de projetos possuindo gráficos de Gantt para auxiliar na representação visual dos projetos e seus prazos de entregas. Foi escrita em Ruby utilizando o framework Ruby on Rails, é multiplataforma e suporta diversos bancos de dados, possibilitando também o uso integrado com vários repositórios tais como Git, Svn, Mercurial, entre outros.

Já o Bugzilla, é uma ferramenta de rastreamento de bugs online desenvolvida pela Fundação Mozilla e lançada em 1998 como um

[6] https://www.bugzilla.org/
[7] https://www.mantisbt.org/
[8] https://www.redmine.org

dos primeiros produtos da Mozilla. Originalmente foi escrita em TCL, mas logo após foi reescrita em Perl. Atualmente é mantido por voluntários, sendo utilizado por centenas de projetos de código aberto ou proprietário, como a Nasa e a Wikipédia, precisando também de um banco de dados SQL e do Apache para poder ser utilizada.

5.1.1 Histórico de Vulnerabilidades. Esta seção apresenta o histórico de vulnerabilidades já encontradas nas ferramentas que serão testadas neste trabalho, levando em consideração as versões lançadas ao longo dos anos segundo o CVE Details, uma base de dados de vulnerabilidades de segurança [1].

Na Figura 1, são apresentadas as vulnerabilidades de segurança encontradas para a ferramenta Redmine. Segundo esse histórico, observa-se que nenhuma vulnerabilidade do tipo *SQL Injection* foi encontrada, ao passo que um alto número de vulnerabilidades de XSS se faz presente nas versões 0.7.2, 0.8.5, 0.8.7, 1.0.1, 1.0.5, 1.3.2, 2.6.2, 3.2.3, 3.2.6, 3.2.8 e 3.3.3 entre os anos de 2008 e 2018.

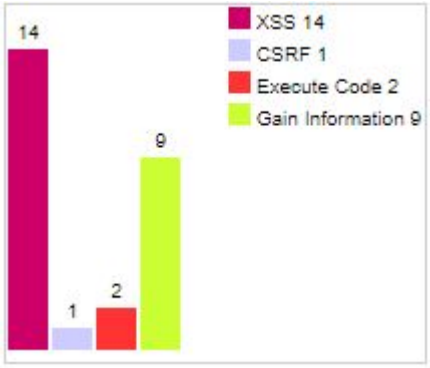

Figura 1: Vulnerabilidades no Redmine entre 2008 a 2018[4]

Para a ferramenta Bugzilla, pode-se observar na Figura 2 um elevado número de vulnerabilidades do tipo XSS encontradas entre as versões 2.14 à 4.5.1, e um número relativamente baixo de *SQL Injection* com base no tempo avaliado pela ferramenta nas versões 2.14, 2.16, 2.17, 2.18, 2.20 e 3.0. Desde 2010, não foram encontradas vulnerabilidades do tipo *SQL Injection* e, nos últimos quatro anos, foram poucas vulnerabilidades do tipo XSS encontradas.

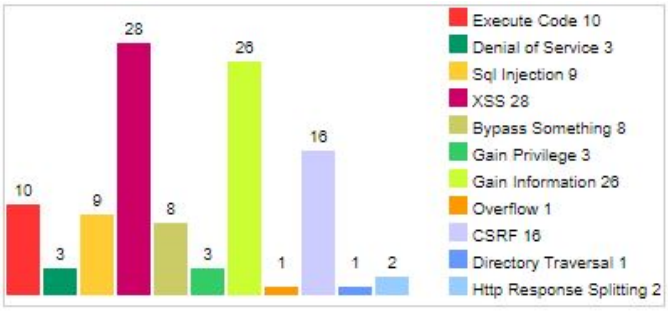

Figura 2: Vulnerabilidades no Bugzilla entre 2000 a 2017[2]

A ferramenta MantisBT também apresentou um elevando número de vulnerabilidades do tipo XSS encontradas nas versões 1.2.2 à 1.2.18, 1.3.0 à 1.3.13, 2.1.0, 2.2.1, 2.2.3, 2.5.2, 2.15.0, sendo em 2017 encontradas quatorze das trinta e seis, o que representa que a ferramenta estava muito vulnerável contra este tipo de ataque. Percebe-se também um número baixo de *SQL Injection* nos últimos dez anos nas versões 1.2.13 à 1.2.19, 1.3.0 e 2.10.0, como mostrado no gráfico do CVE Details [3] na Figura 3.

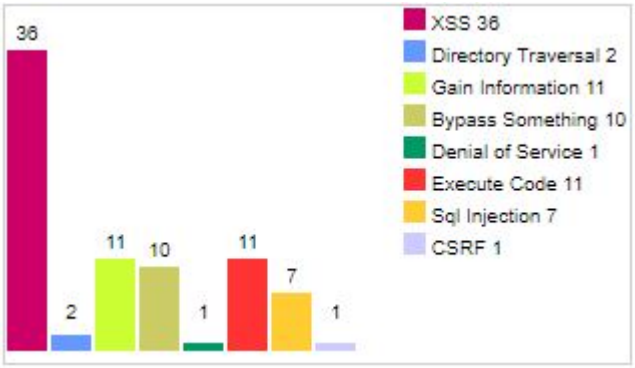

Figura 3: Vulnerabilidades no Mantisbt entre 2008 a 2018[3]

5.2 Preparação dos testes

Esta seção apresenta os cenários e os casos de testes criados, a instalação e a configuração das ferramentas de teste, a preparação do ambiente e a execução dos testes.

5.2.1 Cenários e Casos de Testes. Com base na definição das ferramentas a serem testadas e das vulnerabilidades definidas, foi realizado um estudo para investigar as funcionalidades comuns no contexto de *issue tracking*, com o objetivo de criar os cenários de testes. O estudo baseou-se em como as ferramentas implementam seus mecanismos de criação de *issue*, que foi definido como objetivo principal, pois o Redmine por exemplo, também desempenha outras funções além de issue tracker, como gerenciador de projetos, que não se aplica no contexto deste trabalho.

Para as três ferramentas, serão testadas as seguintes funcionalidades:

- Criação de *Issue* - Esta funcionalidade tem por objetivo criar um *issue*.
- Edição de *Issue* - Esta funcionalidade tem por objetivo editar um *issue* criado.
- Consulta de *Issue* - Esta funcionalidade tem por objetivo consultar um *issue* criado.
- Visualizar um *Issue* - Esta funcionalidade tem por objetivo visualizar as informações contidas em um *issue* criado.
- Modificar um *Issue* - Esta funcionalidade tem por objetivo modificar as informações contidas em um *issue* criado.
- Modificar Privilégios - Objetivo de conseguir alterar privilégios de usuários no sistema.
- Bloqueio de Sessão - Bloqueio de sessão após período de inatividade.
- Acesso Direto à Página - Acessar diretamente uma página sem realizar login no sistema.

- Ferramenta possui Sistema de Logout caso Usuário feche o Navegador Diretamente - Após sair do sistema sem realizar logout, a ferramenta desloga o usuário.
- Páginas Acessadas Anteriormente após Logout - Páginas acessadas anteriormente após logout.
- Força Bruta - Ataques de força buta são realizados com várias combinações de senhas e usuários.
- Protocolos Utilizados - Tipo de protocolo utilizado pelas ferramentas, se protege os dados.
- Métodos de Envio - Utiliza quais métodos para enviar informações dos *issues* criados.

A Tabela 2 ilustra as funcionalidades testadas em cada ferramenta. Ressalta-se que, na ferramenta MantisBT, como não é possível visualizar e modificar *issue* não estando logado no sistema não foi possível realizar o teste de *Autenticação*, já que a ferramenta não apresenta campos de navegação na tela de login.

Tabela 2: Funcionalidades Testadas

Funcionalidades Testadas	Bugzilla	MantisBT	Redmine
Criação de Issue	✓	✓	✓
Edição de Issue	✓	✓	✓
Consulta de Issue	✓	✓	✓
Visualizar Issue	✓	✗	✓
Modificar Issue	✓	✗	✓
Modificar Privilégios	✓	✓	✓
Bloqueio de Sessão	✓	✓	✓
Acesso Direto à Página	✓	✓	✓
Páginas Acessadas Anteriormente após Logout	✓	✓	✓
Ferramenta possui Sistema de Logout caso Usuário feche o Navegador diretamente	✓	✓	✓
Força Bruta	✓	✓	✓
Protocolos Utilizados	✓	✓	✓
Métodos de Envio	✓	✓	✓

Ao todo foram criados treze cenários e casos de teste para testar as funcionalidades selecionadas. A Figura 4 ilustra um cenário de teste definido para a vulnerabilidade de *Controle de Acesso* para a funcionalidade de modificação de um *issue*. Para cada cenário, foram identificadas pré-condições, como usuário não autenticado, issue criado, variando para cada cenário de teste.

Para a elaboração dos casos de testes, a partir dos cenários criados, foi utilizada uma ferramenta de automação de testes chamada Katalon[9], uma ferramenta *open source* desenvolvida pela Katalon LLC e construída sobre a estrutura do Selenium. Essa ferramenta permite controlar a execução dos testes, aumentando a produtividade e reduzindo tempo com a realização de uma mesma tarefa, além de possibilitar a geração de scripts em vários formatos.

A Figura 5 ilustra um trecho de um de caso de teste com base no cenário de teste citado na Figura 4, que foi criado a partir da ferramenta Katalon. Este cenário representa a tentativa de modificar um *issue* não estando logado nas ferramentas sob teste.

[9]https://www.katalon.com/

Descrição: Este cenário de teste tem objetivo de modificar um issue, não autenticado no sistema.

Pré-Condição: Issue criado, usuário não autenticado.

1. Usuário solicita a página de login do sistema.
2. Sistema exibe uma tela com um campo de projetos.
3. Usuário solicita acesso à respectiva página.
4. Sistema exibe uma tela com os projetos presentes.
5. Usuário solicita acesso a um projeto.
6. Sistema exibe uma tela com issue presentes no projeto.
7. Usuário abre um issue.
8. Usuário solicita alteração no issue.

Figura 4: Exemplo de um cenário de teste de modificação de issue

5.2.2 Preparação do ambiente de teste. Esta seção apresenta os detalhes de instalação e configuração das ferramentas sob teste, necessários à execução dos testes.

A versão utilizada do MantisBT foi a 2.16.0, cuja instalação requer instalar uma série de dependências para a linguagem PHP, a ferramenta utiliza um banco de dados e o Apache. Com relação ao Redmine, foi instalada a versão 3.4.6, sendo necessária o banco de dados e uma configuração no Apache. Para o Bugzilla, instalado na 5.0.4, houve a necessidade de instalar dependências para a linguagem Perl, além do MySQL e do Apache. É importante destacar também que as versões instaladas aparentam não terem sido testadas e avaliadas no CVE, pois não foram encontradas em seus históricos de vulnerabilidades (Seção 5.1.1).

A partir das ferramentas instaladas e dos cenários de testes definidos, de acordo com as vulnerabilidades selecionadas, o ambiente de teste foi devidamente preparado, com a realização de alguns cadastros prévios nas ferramentas, tais como, criação de um usuário, criação de um projeto, de uma categoria, um papel e finalmente o *issue*.

5.2.3 Execução dos testes. Para a realização dos testes sobre as vulnerabilidades de *Cross Site Scripting* e *SQL Injection* inicialmente foi necessário gerar os casos de testes a partir dos cenários definidos. Para gravar os passos de cada cenário de teste, a ferramenta Katalon foi usada. Além disso, para que as ferramentas sob teste funcionassem adequadamente, o servidor *Web Apache* e o MYSQL foram habilitados e inicializados.

A partir dos casos de testes gerados, a ferramenta *Owasp Zap* foi inicializada. É importante destacar que para a execução da ferramenta, foi necessário configurar a porta do navegador para que as páginas salvas na ferramenta Katalon pudessem ser capturadas e usadas na ferramenta *Owasp Zap* e assim realizar os testes.

A *Owasp Zap* realiza alguns testes automáticos pré-definidos em uma janela chamada varredura ativa. Dentre esses testes, tem-se os três tipos de *Cross Site Scripting* e *SQL Injection* mencionados na seção 2 que foram executados nas ferramentas de *issue tracking* selecionadas com base nos cenários de testes definidos. Alguns testes foram cancelados pelo motivo de não fazerem parte das vulnerabilidades selecionadas neste trabalho.

```
@Before
public void setUp() throws Exception {
  driver = new FirefoxDriver();
  baseUrl = "https://www.katalon.com/";
  driver.manage().timeouts()
  .implicitlyWait(30, TimeUnit.SECONDS);
}

@Test
public void testUntitledTestCase() throws Exception {
  driver.get("http://localhost:3000/");
  driver.findElement(By.linkText("Projetos")).click();
  driver.findElement(By.linkText("Projeto de Conclusão de Curso")).click();
  driver.findElement(By.linkText("1")).click();
  driver.findElement(By.linkText("Erro na tela de login")).click();
}
```

Figura 5: Trecho do código de um caso de teste de modificação de *issue*

Para a realização dos testes sob a vulnerabilidade de *Criptografia*, foi realizada uma análise na própria ferramenta *Owasp Zap*, pois ela mostra cada protocolo e método de envio utilizado por cada página quando executa a varredura ativa.

Os testes sob as vulnerabilidades de *Controle de Acesso* e *Autenticação*, foram realizados de forma manual, pois a ferramenta *Owasp Zap* não dá suporte a testes de *Controle de Acesso*. Para testar a vulnerabilidade de *Autenticação* há necessidade de estender a ferramenta com scripts, o que é mais trabalhoso de fazer. Como os cenários definidos são simples, a execução manual dos testes sobre essas vulnerabilidades foi realizada sem muito esforço.

5.3 Resultados, Análises e Discussões dos Testes

Nesta seção, são discutidos os resultados dos testes de segurança realizados nas ferramentas de *issue tracking* selecionadas.

A Tabela 3 mostra os resultados dos testes de *Autenticação* realizados nas ferramentas selecionadas, sendo possível verificar que a Bugzilla apresentou-se segura nestes testes, mas é importante ressaltar que quando é feita a requisição de voltar à página anterior na ferramenta, ela permite visualizar a página anterior, só após uma solicitação qualquer na página que ela redireciona para a página de login, sendo que a página retornada pode conter informações que não deveriam ser vistas. Já as ferramentas MantisBT e Redmine apresentaram uma falha de permissão com ataques de força bruta, o que favorece ao atacante um ataque com milhares de combinações de logins e senhas. O MantisBT apresenta uma tela para login e uma outra para o campo de senha o que representa um grau a mais de dificuldade ao atacante, embora pouco eficiente contra estes tipos de ataques.

Os testes realizados sobre a vulnerabilidade de *Criptografia* apresentados na Tabela 4 mostram que o tipo de protocolo utilizado não é seguro, ou seja, não se utiliza um protocolo que criptografe o que está sendo enviado pela rede. Essa falha é crítica, sendo um grande problema contra ataques em rede, nos quais são realizados tentativas de interceptar e obter estas informações na rede. Por outro lado, as informações são enviadas pelo método POST que protege as informações inseridas em campos de formulários, sendo um ponto positivo para as três ferramentas testadas. Mas essa combinação

Tabela 3: Resultados do Teste de Autenticação

Ferramentas	Permite acesso direto à página	Permite páginas acessadas após logout	Permite ataques de força bruta
MantisBT	✗	✗	✓
Redmine	✗	✗	✓
Bugzilla	✗	✗	✗

HTTP e POST é perigosa já que em um formulário com um script malicioso é possível que os dados sejam enviados automaticamente sem a vítima saber, assim o ideal é utilizar o protocolo HTTPS juntamente com o método POST em campos de formulários.

Tabela 4: Resultados do Teste de *Criptografia*

Ferramentas	Protocolo Utilizado	Método de envio (GET e POST)
MantisBT	HTTP	POST
Redmine	HTTP	POST
Bugzilla	HTTP	POST

As Tabelas 5 e 6 ilustram as vulnerabilidades de *SQL Injection* e *Cross Site Scripting* encontradas pela ferramenta *Owasp Zap*, destacando quais ferramentas apresentaram mais vulnerabilidades em suas respectivas versões instaladas. Estas vulnerabilidades estão classificadas pelo grau de risco, sendo: A - risco alto, M - risco médio, e B - risco baixo.

A partir da Tabela 5 para a vulnerabilidade de *SQL Injection*, nota-se que na ferramenta Redmine foi encontrada somente uma vulnerabilidade considerada de alto risco. Considerando que, segundo o histórico de vulnerabilidades conhecidas para a ferramenta (Seção 5.1.1), não haviam vulnerabilidades deste tipo, a ferramenta mostra-se frágil para este tipo de ataque. Na ferramenta Mantis Bug Tracker, foram encontrados quatro tipos dessa mesma vulnerabilidade, com alto risco, o que representa um número alto, considerando-se o histórico desta ferramenta (Seção 5.1.1). Por fim, na ferramenta

Bugzilla, foram encontradas três ocorrências da vulnerabilidade citada, que representa um número alto já que desde de 2010 não foram encontradas vulnerabilidades deste tipo na ferramenta.

Tabela 5: Resultados do Teste de *SQL Injection*

Cenários de testes	Bugzilla	MantisBT	Redmine
Criação de Issue	A	A	-
Edição de Issue	A	A	-
Consulta de Issue	-	A	A

A Tabela 6 informa que, dentre as vulnerabilidades de *Cross Site Scripting* encontradas, apenas uma foi classificada de alto risco para a ferramenta Bugzilla. A ferramenta Redmine não apresentou nenhuma vulnerabilidade desse tipo, o que mostra também que medidas corretivas e preventivas estão sendo tomadas na mesma. Na ferramenta Mantis Bug Tracker, foi encontrada a mesma vulnerabilidade, mas de categoria baixa para todos os cenários testados. Isso mostra uma diminuição significativa para essa ferramenta, levando-se em consideração o alto número dessas vulnerabilidades encontradas anteriormente (Seção 5.1.1).

A Tabela 7 ilustra as falhas do tipo *Controle de Acesso* idenficadas nas ferramentas. Verifica-se que as permissões de visualização, modificação de issue e de gerenciamento de privilégios não foram encontrados no MantisBT, pois ele possui uma interface limpa, com apenas os campos de login e senha. Pode-se verificar também que tanto o MantisBT quanto o Redmine possuem bloqueio de sessão e logout automático, ao contrário do Bugzilla. Um usuário pode se esquecer de sair do sistema, por exemplo, e então se um outro usuário abrir o sistema na mesma máquina ele vai ter acesso indevidos aos *issues*.

6 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho apresentou uma avaliação em ferramentas *web open source* de *issue tracking* com base em testes de segurança que avaliaram algumas das vulnerabilidades destacadas pela comunidade Owasp. Os testes foram aplicados sobre as ferramentas Bugzilla, MantisBT e Redmine por serem bastante populares.

Segundo os relatórios de testes, a ferramenta Bugzilla, por estar no mercado há mais tempo e ser uma ferramenta consolidada, apresentou um número alto de vulnerabilidades consideradas de alto risco. A ferramenta Mantis Bug Tracker também apresentou um número elevado de vulnerabilidades de alto risco destacando-se a vulnerabilidade do tipo *SQL Injection*. Já na ferramenta Redmine, foram encontrados um número menor de vulnerabilidades comparando com as outras ferramentas.

Considerando-se que são ferramentas amplamente utilizadas em projetos de software e que tais vulnerabilidades de segurança colocam em risco os projetos controlados pelas ferramentas, ressalta-se a importância das correções necessárias serem aplicadas com urgência.

A partir dos resultados obtidos, pode-se considerar o aspecto de segurança como um fator de escolha para a adoção de tais ferramentas. Segundo as vulnerabilidades selecionadas e os testes realizados com a ferramenta *Owasp Zap*, a ferramenta que apresentou o menor número de vulnerabilidades de segurança dentre as três avaliadas foi a ferramenta Redmine, seguida pelas ferramentas Bugzilla e MantisBT.

Como trabalhos futuros sugere-se que os testes de segurança sejam realizados em outras ferramentas que estejam constantemente presentes no top 10 das ferramentas de *issue tracking* mais utilizadas. Além disso, outras vulnerabilidades de segurança e outras funcionalidades das ferramentas não tratadas aqui, podem ser analisadas.

REFERÊNCIAS

[1] 2009. CVE-CVEDETAILS-the ultimate security vulnerability datasource. https://www.cvedetails.com/

[2] 2009. CVE-product-bugzilla. https://www.cvedetails.com/product/785/?q=bugzilla

[3] 2009. CVE-product-mantisbt. https://www.cvedetails.com/product/19885/?q=Mantisbt

[4] 2009. CVE-product-redmine. https://www.cvedetails.com/product/15112/?q=Redmine

[5] 2015. CVE-product-mantisbt- CVE-2014-8987. https://www.cvedetails.com/cve/CVE-2014-8987/

[6] 2017. CVE-product-bugzilla-CVE-2015-8477. https://www.cvedetails.com/cve/CVE-2015-8477/

[7] 2018. 15 Best Bug Tracking Software: Top Defect/Issue Tracking Tools of 2018. https://www.softwaretestinghelp.com/popular-bug-tracking-software/

[8] Canaltech. 2014. Mozilla Corrige Falha De Segurança Crítica No BugZilla. https://canaltech.com.br/navegadores/Mozilla-corrige-falha-de-seguranca-critica-no-BugZilla/

[9] José Victor Pereira Costa. 2017. Análise de Vulnerabilidades de Segurança em Portais de Governos Eletrônicos. (dezembro 2017).

[10] Lucas de Almeida Lins. 2017. Avaliação de Scanners de Vulnerabilidades-Dez riscos mais críticos em aplicações Web. (julho 2017).

[11] Túlio Spuri Teixeira de Oliveira. 2012. Testes de Segurança em Aplicações Web Segundo a Metodologia Owasp. (novembro 2012).

[12] Ederson Melo. 2018. 20 Best Bug Tracking Software: Top Issue/Defect Tracking Tools of 2018. https:https://www.guru99.com/top-20-bug-tracking-tools.html

[13] Open Web Application Security Project. 2014. *OWASP Open Web Application Security Project.* CreateSpace Independent Publishing Platform, USA.

[14] Open Web Application Security Project. 2014. *OWASP Top 10: The Top 10 Most Critical Web Application Security Threats Enhanced with Text Analytics and Content by PageKicker Robot Phil 73.* CreateSpace Independent Publishing Platform, USA.

[15] Open Web Application Security Project. 2014. *OWASP Zed Attack Proxy Project.* CreateSpace Independent Publishing Platform, USA.

Tabela 6: Resultados do Teste de *Cross Site Scripting*

Cenários de testes	Tipo de XSS	Bugzilla	MantisBT	Redmine
Criação de Issue	Refletido	-	B	-
	Armazenado	-	-	-
	Baseado em DOM	-	-	-
Edição de Issue	Refletido	A	B	-
	Armazenado	-	-	-
	Baseado em DOM	-	-	-
Consulta de Issue	Refletido	-	B	-
	Armazenado	-	-	-
	Baseado em DOM	-	-	-

Tabela 7: Resultados do Teste de *Controle de Acesso*

Ferramentas	Permite visualização de issue	Permite modificação de issue	Permite gerenciamento de privilégios	Possui bloqueio de sessão	Possui sistema de logout automático
MantisBT	✗	✗	✗	✓	✓
Redmine	✓	✗	✗	✗	✓
Bugzilla	✓	✗	✗	✗	✗