Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Thiago Zaninotti

Método para testes de segurança em aplicações web por meio de casos de uso

São Paulo
2012

Thiago Zaninotti

Método para testes de segurança em aplicações web por meio de casos de uso

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT, como parte dos requisitos para a obtenção do título de Mestre em Engenharia da Computação.

Data da aprovação ___/_____/____

_____

Prof. Prof. Dr. Adilson Eduardo Guelfi (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Membros da Banca Examinadora:

Prof. Dr. Adilson Eduardo Guelfi (Orientador)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dr. Alexandre Jose Barbieri de Sousa (Membro)
IPT – Instituto de Pesquisas Tecnológicas do Estado de São Paulo

Prof. Dr. Roberto Kenji Hiramatsu (Membro)
USP – Universidade de São Paulo

Thiago Zaninotti

Método para testes de segurança em aplicações web por meio de casos de uso.

Dissertação de Mestrado apresentada ao Instituto de Pesquisas Tecnológicas do Estado de São Paulo – IPT, como parte dos requisitos para obtenção do título de Mestre em Engenharia da Computação.

Área de Concentração: Redes de Computador.

Orientador: Prof. Dr. Adilson Eduardo Guelfi

São Paulo
Abril/2012

Ficha Catalográfica
Elaborada pelo Departamento de Acervo e Informação Tecnológica – DAIT
do Instituto de Pesquisas Tecnológicas do Estado de São Paulo - IPT

Z37m **Zaninotti, Thiago**
Método para testes de segurança em aplicações web por meio de casos de uso. / Thiago Zaninotti. São Paulo, 2012.
95p.

Dissertação (Mestrado em Engenharia de Computação) - Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Adilson Eduardo Guelfi

1. Teste de segurança 2. Aplicações Web 3. Protocolo http 4. SDLC 5. SCOUT 6. Tese I. Instituto de Pesquisas Tecnológicas do Estado de São Paulo. Coordenadoria de Ensino Tecnológico II. Título

12-36 CDU 004.49'738.52(043)

DEDICATÓRIA

Dedico este trabalho aos meus pais, José Carlos e Márcia, cujos esforços e incentivos me proporcionaram todo o arcabouço necessário para a conclusão da minha dissertação.

AGRADECIMENTOS

Agradeço a paciência e atenção do Prof. Mário Miyake, coordenador do curso, que apoiou o desenvolvimento do tema da dissertação, mesmo quando os prazos estavam tão apertados.

Ao meu orientador Prof. Adilson Guelfi que, apesar dos curtos prazos, acreditou no tema e ajudou-me a transformar esta idéia em dissertação de mestrado.

Aos professores membros da banca pela atenção, tempo e conselhos importantes para a finalização deste documento.

Aos professores que me aconselharam durante o processo de consolidação do meu tema, em especial ao Prof. Marcelo Rezende, Prof. Antonio Rigo e Prof. Edite Lino de Campos.

Por final, agradeço a todos da equipe da secretaria do IPT pela paciência e assistência nestes últimos meses de esforços empreendidos até a defesa da dissertação.

RESUMO

Os esforços empreendidos atualmente para melhorar a segurança dos aplicativos web por meio da consolidação de modelos de maturidade ainda carecem de eficácia para gestão de vulnerabilidades, principalmente com relação à ausência da premissa do conhecimento prévio das funcionalidades do sistema nos testes de segurança. Trabalhos recentes ressaltam a importância desta premissa, contudo, existem algumas restrições que podem servir como oportunidade para implantação de método complementar, incluindo: (a) Uso de técnicas independentes da leitura do código-fonte; (b) Utilização de interações específicas com base nas funcionalidades dos aplicativos; (c) Enfoque na integração ao SDLC pela repetição sistemática dos testes e (d) emprego da característica de extensibilidade para permitir a detecção de um número maior de vulnerabilidades. Este trabalho apresenta um método para execução de testes de segurança por meio de casos de uso, permitindo a sua aplicação no contexto do aplicativo web. A principal característica do método é o requisito de utilização de casos de uso que configurem funcionalidades específicas do aplicativo. A contribuição deste trabalho é aumentar a eficácia na detecção de vulnerabilidades, apoiar a utilização sistemática das atividades de teste ao longo do SDLC, proporcionar extensibilidade para implantação de novas classes de ataques e permitir a aceleração da adoção de modelos formais de segurança. A validação da proposta é realizada por meio de testes comparativos em relação aos outros métodos estudados com a aplicação *wordpress*. Emprega-se o critério de assertividade na detecção de vulnerabilidades existentes em cinco casos comuns de uso deste aplicativo. Após os testes, o SCOUT apresenta um aumento de 20% na eficácia de detecção para as transações que exigem conhecimento prévio de funcionalidades.

Palavras Chaves: testes de segurança; aplicações web; SDLC; gestão de vulnerabilidades; casos de uso.

ABSTRACT

Method for testing web application security through use cases

Efforts currently undertaken to improve the security of web applications through consolidation of maturity models still lack efficiency for vulnerability management, particularly with respect to the premise of no prior knowledge of system functionality for security testing. Recent studies underscore the importance of this premise, however, there are some restrictions that can serve as an opportunity to develop a supplementary method, including: (a) Use of techniques that do not rely on source code reading; (b) Use of specific interactions based on application functionalities; (c) Focus on integration into SDLC by means of systematic repetition of tests and (d) the use of extensibility to allow detection of a greater number of vulnerabilities. This paper presents a method for performing security tests through use cases, allowing its application in the context of web application. The main feature of the method is the requirement to utilize use cases to configure application-specific functionalities. The contribution of this work is to increase the effectiveness in detecting vulnerabilities, support the use of systematic testing activities throughout the SDLC, providing extensibility to implement new classes of attack and allow the acceleration of the adoption of formal models of security. The proposal validation is being performed using the wordpress application by means of comparative tests in relation to other methods studied. It employs the criterion of assertiveness in detecting vulnerabilities throughout five common use cases for this application. After testing, the SCOUT has a 20% increase in detection efficiency for transactions that require prior knowledge of features.

Keywords: security tests; web applications; SDLC; vulnerability management; use cases.

Lista de Ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

ASVS	*Application Security Verification Standard*
BSI-MM	*The Building Security in Maturity Model*
CVE	*Common Vulnerabilities Exposures*
DoD	*US Department of Defense*
HTTP	*Hypertext Transfer Protocol*
NIST	*National Institute of Standard Technology*
NVD	*National Vulnerability Database*
OSVDB	*Open Source Vulnerabilities Database*
OWASP	*The Open Web Application Security Project*
REST	*Representational State Transfer*
SDLC	*Secure Development Life-Cycle*
SPI	*Software Protection Initiative*
URI	*Uniform Resource Information*
URL	*Uniform Resource Location*

Sumário

1 INTRODUÇÃO

1.1 Contexto do trabalho

Com o advento de plataformas de desenvolvimento orientadas à interpretação em tempo de execução (*script languages*) e o uso intensivo da computação distribuída, a utilização de aplicativos web tornou-se uma prática disseminada para implantação de componentes operacionais básicos da sociedade, incluindo sistemas de saúde, impostos, serviços públicos, financeiros e comércio eletrônico (GLISSON et al., 2007).

Contudo, a adoção do uso de aplicativos web pressupõe a implantação progressiva de arquiteturas abertas mais complexas de desenvolvimento, incluindo aquelas orientadas a serviços, produzindo um terreno fértil para a exploração de vulnerabilidades de segurança.

1.1.1 Cenário atual das vulnerabilidades dos aplicativos web

De acordo com Shahriar e Zulkernine (2009), a prática de desenvolver aplicações mais seguras foi estabelecida há mais de uma década. Desde então, técnicas como análise estática de códigos-fontes, análise dinâmica em tempo de execução e proteção automática contra falhas, têm sido empregadas para identificar e prevenir vulnerabilidades. Entretanto, apesar da ampla disponibilidade destas práticas, falhas de segurança em aplicações web em pleno estágio de produção continuam sendo reportadas com frequência por bancos de dados públicos como *Common Vulnerabilities and Exposures* (CVE)*, Open Source Vulnerabilities Database* (OSVDB) e *Bugtraq.*

Neste sentido, Assad et al. (2010) também discorrem sobre o crescimento significativo das ameaças de segurança nos sistemas, destacando, principalmente, a introdução de novos requisitos na construção de aplicativos web, incluindo alta interatividade e exposição de interfaces de negócio pelo uso de serviços, como um grande fator de contribuição para o aumento da complexidade e vulnerabilidades das aplicações.

Em uma pesquisa conduzida recentemente pela empresa Cenzic (2010), 73% das grandes organizações entrevistadas admitiram, nos últimos 24 meses, ter sofrido incidente de segurança por meio da exploração de vulnerabilidades em seus aplicativos web. Segundo o relatório, mesmo após os incidentes, 85% destes aplicativos web ainda possuíam ao menos uma vulnerabilidade crítica.

1.1.2 Cenário dos modelos de segurança para aplicativos web

Para mudar o cenário das vulnerabilidades, organizações como Microsoft, Cigital e *The Open Web Application Security Project* (OWASP) têm proposto enfrentar o problema por meio da implantação de requisitos formais de segurança durante todo o ciclo de processos de construção e gestão de aplicativos web.

Estas organizações recomendam, na elaboração de modelos de maturidade que estabelecem práticas comuns para gestão do ciclo de desenvolvimento seguro (SDLC), o emprego ostensivo de atividades de testes de segurança por todo o ciclo de vida da aplicação, garantindo, assim, a eficácia dos seus mecanismos de proteção (GRAHAM et al. 2006).

Entretanto, apesar da ampla disponibilidade destes modelos e da importância das atividades de teste de segurança, os resultados de um estudo realizado pelo *The Building Security in Maturity Model* (2009) demonstram que apenas um terço das maiores empresas de tecnologia dos Estados Unidos realiza formalmente atividades de teste integradas ao ciclo de desenvolvimento de software.

Acredita-se que esta deficiência, além de refletir os dados apresentados pela Cenzic (2010), possa ser explicada pelos resultados obtidos pela pesquisa conduzida pela empresa Errata em 2010. O estudo aponta que, apesar de 80% das equipes de desenvolvedores reconhecerem a existência de modelos de maturidade, apenas 30% implementam atividades formais de segurança, incluindo testes de vulnerabilidade. Cerca de 40% dos entrevistados apontaram a falta de uma janela de tempo adequada e, principalmente, a falta de recursos técnicos, incluindo métodos, ferramentas e equipe, como a principal causa da dificuldade para adesão (GEER, 2010).

Desta forma, infere-se que, apesar da existência de modelos de segurança que especifiquem requisitos formais para testes de segurança integrados ao ciclo de desenvolvimento, estas atividades carecem de ampla implantação no SDLC, seja pelo desconhecimento, falta de recursos técnicos ou falta de tempo, dificultando a gestão eficaz de vulnerabilidades.

1.1.3 Cenário atual dos testes de segurança

De acordo com o *The Building Security in Maturity Model* (BSI-MM), cerca de 90% das empresas pesquisadas optaram por implantar atividades complementares aos testes de segurança, incluindo os testes de invasão ou *penetration-tests.* Estes, realizados independente das atividades de desenvolvimento, têm por característica executar, em um curto espaço de tempo, uma análise superficial da aplicação antes da sua promoção para a produção (MCGRAW et al., 2010).

Segundo Thompson (2003), o teste de invasão consiste em utilizar ataques conhecidos anteriormente e, por meio de uma base de conhecimento destas vulnerabilidades, aplicá-lo contras as novas aplicações para aferir o possível comportamento positivo de um ataque. O problema é que, uma vez que os ataques evoluem, o teste passa a ter eficácia restrita com o passar do tempo. Thompson acredita que o sucesso para um bom teste estaria em traduzir as técnicas de ataque para testes práticos para que fossem utilizados nos casos de uso específicos do aplicativo, monitorando-se todas as interações entre o usuário e os recursos computacionais da aplicação.

Em recente pesquisa, Assad et al. (2010) também faz alusão ao problema dos testes atuais. Constata-se que uma das formas eficazes de encontrar vulnerabilidades seria entender em que parte da aplicação as falhas poderiam estar para, então, adaptar os testes de segurança às suas funcionalidades. Desta forma, uma vez que a maioria dos testes realizados utilizam meramente uma biblioteca de ataques conhecidos e nenhum conhecimento prévio do sistema (BSI-MM, 2009), eles não seriam eficazes para detectar e prevenir vulnerabilidades, o que explicaria o crescimento contínuo das falhas de segurança ao longo do tempo (SHAHRIAR; ZULKERNINE, 2009).

Infere-se, portanto, que um dos problemas relacionados com o cenário atual das vulnerabilidades nos aplicativos web está ligado com o pouco ou nenhum uso do conhecimento prévio das funcionalidades do sistema. Esta questão, aliada aos problemas já discutidos no item 1.2, como ausência de janela de tempo e recursos técnicos, estariam dificultando a elaboração de requisitos mais eficientes de segurança e, consequentemente, testes mais eficazes para detecção de vulnerabilidades (ASSAD et al., 2010).

1.2 Motivação

Em um estudo comparativo de métodos de detecção de vulnerabilidades, Shahriar e Zulkernine (2009) levantaram 20 propostas práticas para o tema e, apesar de apenas 40% tratarem especificadamente do ambiente de aplicativos web, a maioria destes métodos tem por requisito o conhecimento prévio das funcionalidades do aplicativo para realização dos testes de segurança.

Entretanto, apesar da característica positiva de utilização dos pontos de entrada de dados e interação do aplicativo para a realização de casos de teste de segurança, ainda persistem algumas limitações que, de acordo com os cenários descritos anteriormente, podem restringir a adoção e eficácia dos testes de segurança no contexto atual dos aplicativos web, incluindo:

a) Métodos com processos mais maduros e amplo espectro de ferramentas, tais como o de testes caixa-preta (BAU et al., 2010) ou testes caixa-branca (MCGRAW; POTTER, 2004), geralmente não utilizam o conhecimento prévio das funcionalidades do sistema ou não contemplam a repetição dos testes ao longo das diferentes fases do SDLC;

b) Métodos que permitem a descoberta automática de características prévias do sistema, tais como WAVES (HUANG et al., 2004), SecuBat (KALS et al., 2006) e D-WAV (ZHANG et al., 2010) não permitem a utilização de um fluxo de atividades passo-a-passo que caracterize um caso de uso do aplicativo;

c) Métodos que permitem a criação de testes de segurança por meio de casos de uso, como ARDILLA (KIEZUN et al., 2006), necessitam de acesso ao código-fonte do aplicativo, uma opção cada vez mais restritiva para a realidade heterogênea das aplicações web (CANFORA; DI PENTA, 2006);

d) Com exceção do método de teste caixa-preta, todos os métodos estudados implementam espectro reduzido de classes de vulnerabilidades, restringindo a extensibilidade dos testes de segurança.

Desta forma, a motivação deste trabalho se sustenta nos seguintes fatores específicos, inferidos a partir das limitações acima:

a) A oportunidade de desenvolver um método que permita a utilização de casos de uso funcionais do aplicativo web para a construção de testes de segurança, sem a necessidade da leitura do código-fonte como requisito de entrada;

b) A possibilidade de utilização de conhecimentos prévios do aplicativo web, por meio de casos de uso, que permitam maior interação e acesso controlado às suas transações, sobrepondo a restrição da descoberta automática de URL;

c) A lacuna existente nos trabalhos recentes com relação à integração do método de teste de segurança ao SDLC, permitindo que os testes sejam sistematicamente repetidos e comparáveis nas diferentes fases do ciclo de vida;

d) Possibilidade de aumentar o espectro dos casos de teste para a detecção de um número maior de classes de vulnerabilidades.

Entende-se que este trabalho deve abordar estes fatores com o objetivo de aumentar a assertividade dos testes de segurança, colaborando, desta forma, com a otimização do processo e a resolução dos problemas descritos no contexto do trabalho, incluindo questões de falta de recursos técnicos e falta de janela de tempo para detecção de vulnerabilidades.

1.3 Objetivo

Este trabalho tem por objetivo propor um método para execução de testes de segurança por meio de casos de uso, a ser aplicado para o contexto de aplicativos web. O método proposto neste trabalho será referenciado pelo acrônimo SCOUT, que representa a expressão *Security Casting through Online Use case Testing.*

1.4 Escopo do trabalho

As seguintes atividades estão contempladas no escopo do trabalho:

a) Atividade 1: Pesquisa de conceitos relevantes para o referencial teórico do trabalho;

b) Atividade 2: Pesquisa e leitura de trabalhos relacionados para o estado da arte;

c) Atividade 3: Definição das fases e atividades do método de teste de segurança por meio de casos de uso;

d) Atividade 4: Definição do ambiente de validação e especificação de requisitos funcionais ferramenta de testes;

e) Atividade 5: Configuração do ambiente experimental para os testes de validação;

f) Atividade 6: Realização dos testes de segurança no ambiente experimental;

g) Atividade 7: Coleta e análise de resultados dos testes para validação dos objetivos do trabalho;

h) Atividade 8: Elaboração e revisão da dissertação;

i) Atividade 9: Escrita e submissão de artigo sobre o método proposto;

j) Atividade 10: Defesa da dissertação.

1.5 Método do trabalho

Os principais métodos utilizados neste trabalho são:

- Estudo dos conceitos relevantes aos testes de segurança e vulnerabilidades dos aplicativos web para estabelecer referencial teórico para o trabalho;
- Pesquisa bibliográfica para levantamento dos principais trabalhos existentes na área, incluindo os métodos relacionados que possam ser utilizados para fins comparativos;
- Análise experimental do método proposto para testes de segurança por meio da elaboração de uma ferramenta que permita validação dos objetivos do trabalho em um contexto específico;
- Coleta de resultados da análise experimental, incluindo a utilização de outros métodos inseridos no contexto do estado da arte, para obter conclusões sobre sua eficácia e aplicabilidade de acordo com critérios pré-estabelecidos.

1.6 Contribuições esperadas

As principais contribuições que este trabalho pretende atingir são:

- Colaborar com o aumento da assertividade na detecção de vulnerabilidades em aplicativos web, incluindo a redução de falsos-positivos e falsos-negativos;
- Proporcionar melhor integração dos testes de segurança no SDLC, permitindo que sua aplicabilidade seja contínua durante todo o ciclo de vida do aplicativo;
- Proporcionar um método genérico que seja independente de classes de vulnerabilidades e permita a extensibilidade para quaisquer tipos de casos de teste;
- Acelerar a adoção dos modelos de maturidade de segurança dos aplicativos web e suas atividades formais de testes de segurança.

1.7 Organização da dissertação

Este trabalho está organizado da seguinte forma:

- No capítulo 2, Referencial teórico, são apresentados os conceitos relevantes relacionados à proposta do trabalho;
- O capítulo 3, Estado da arte dos testes de segurança, apresenta o estado da arte em que este trabalho se encontra inserido, incluindo um panorama dos trabalhos relacionados, bem como suas principais vantagens e desvantagens;
- O capítulo 4, Método para testes de segurança em aplicações web, trata da descrição do método SCOUT proposto neste trabalho, detalhando as fases e atividades para sua execução em testes de segurança, bem como a proposição para validação dos objetivos do trabalho;
- O capítulo 5, Validação do trabalho, contém a especificação do roteiro para validação do método SCOUT, bem como descrições do ambiente experimental para testes, configuração e os dados coletados;
- No capítulo 6, Conclusão do trabalho, é apresentada a análise dos resultados do trabalho em comparação com os métodos estudados no estado da arte.

2 REFERENCIAL TEÓRICO

Esta seção tem por objetivo apresentar o referencial teórico para o melhor entendimento deste trabalho.

Serão apresentados os conceitos de falhas de software, vulnerabilidade e testes de segurança em aplicações web. Posteriormente tratar-se-á dos casos de uso, sua representação e relacionamento com os testes nos aplicativos web. Também serão discutidos os mecanismos para troca de dados e controle de sessão nos aplicativos web.

2.1 Falhas de software e vulnerabilidades nos sistemas

De uma maneira geral, falhas de software são erros de programação que provocam um comportamento inesperado ou incorreto em um sistema. Quando estas deficiências, entretanto, permitem acesso ou uso não autorizado, trata-se de uma vulnerabilidade na segurança do sistema (HOWARD; MEUNIER, 2002).

Para a Iniciativa de Proteção de Software (SPI) do Departamento de Defesa dos Estados Unidos (DoD*)*, a vulnerabilidade de um sistema pode ainda ser definida pela intersecção da ocorrência, acesso e capacidade de explorar falhas.

Em conjunto com as falhas de projeto e operação, as falhas de software são uma das principais causas da ocorrência de vulnerabilidades nos sistemas, entretanto, trata-se dos problemas mais difíceis de serem detectados, uma vez que se exige uma interação de maneira diferente daquela para qual o sistema foi projetado inicialmente (ARKIN et al., 2005).

As principais categorias de falhas de software que causam vulnerabilidades de segurança são:

a) Violações de memória, que permitem a execução arbitrária de instruções ou vazamento de informações;

b) Erros na validação de dados de entrada, que consistem em utilizar dados recebidos por um usuário sem a validação devida de consistência;

c) Extensão de privilégios, que permite a utilização arbitrária de determinadas funcionalidades do sistema buscando-se privilégios maiores do que aqueles que normalmente seriam concedidos.

2.2 Testes de segurança

De acordo com o *National Institute of Standard Technology* (NIST), teste de segurança é um processo para determinar se um sistema possui medidas de proteção adequadas para suas funcionalidades e dados. Quando aplicado em uma situação em particular de um aplicativo, este processo é chamado de caso de teste.

Diferentemente do conceito clássico da engenharia de software que estabelece que os testes de software sejam interações elaboradas para assegurar que um código comporte-se da maneira para qual ele foi projetado (MYERS, 2004), os testes de segurança são um conjunto de técnicas que permite detectar vulnerabilidades por meio de um comportamento intencional distinto da finalidade inicial do software (ARKIN et al., 2005).

Por meio de estudo publicado, Shahriar e Zulkernine (2009) concluem que o uso de técnicas para o teste de segurança de aplicativos é uma prática conhecida há mais de uma década. Eles classificam os métodos mais comuns para detecção de vulnerabilidades de acordo com as técnicas empregadas:

a) Testes de análise estática, que compreendem a análise do código-fonte por meio da varredura em busca de padrões inseguros de codificação;

b) Testes de análise dinâmica e comportamental, que compreendem a identificação de vulnerabilidades em tempo de execução por meio de interações com os pontos de entrada de dados;

c) Testes de monitoração, também conhecidos como sistema de proteção automática contra vulnerabilidades, que funciona por meio

de monitoração do uso da aplicação e a interferência no fluxo de execução quando identificado um padrão suspeito.

2.2.1 Testes de Segurança em Aplicativos web

Um aplicativo web é um software de computador projetado para executar um conjunto de tarefas com a finalidade de assistência ao usuário por meio de um servidor web, que, por sua vez, baseia-se no paradigma do protocolo *Hypertext Transfer Protocol* (HTTP) para a comunicação com o cliente (navegador web). De uma maneira genérica, o navegador web envia um pedido para o servidor web que, então, transmite tal pedido para o aplicativo web. Este realiza as tarefas solicitadas no pedido e retorna uma resposta ao cliente geralmente sob a forma de um documento baseado em linguagem de hipertexto (HTML).

Os métodos utilizados para testar a segurança de aplicativos web podem ser classificados de maneira semelhante à proposta de Shahriar e Zulkernine (2009), ou seja, a partir da análise de seu código-fonte, análise em tempo de execução ou monitoração passiva de seu comportamento. Entretanto, de acordo com Canfora e Di Penta (2006), aplicativos web estão cada vez mais complexos e distribuídos, característica que impõe condições restritivas para acesso completo ao código-fonte e a realização de testes de análise estática.

2.2.2 Características das análises dinâmicas para aplicativos web

De acordo com Michael e Radosevich (2009), os testes dinâmicos de segurança para aplicativos web geralmente possuem as seguintes características:

a) São executados por um agente que possui conhecimento limitado do sistema a ser testado, sem acesso ao código-fonte e entendimento de suas funcionalidades. Em muitos casos, entende-se que o único requisito para os testes é o endereço da aplicação, conhecido como *Uniform Resource Location* (URL);

b) São geralmente apoiados por uma ferramenta de testes conhecida como caixa-preta ou *black-box*, que permite automatizar e gerenciar as análises;

c) Exigem conhecimento técnico especializado em ataques de aplicativos web.

2.2.3 Protocolo HTTP nos testes de segurança

Qualquer método para testes de segurança de aplicativos web que envolva o uso de análise dinâmica deverá realizar interações em tempo de execução por meio do protocolo HTTP.

Trata-se de um protocolo para transferência de dados entre o cliente (navegador web) e o servidor (servidor web) em formato de pergunta e resposta. De acordo com seus projetistas, o protocolo foi desenvolvido para ser genérico; flexível e sem estado de sessão, com o objetivo de simplificar o papel do servidor e permitir um reduzido consumo de recursos limitado às transações correntes.

Inicialmente utilizado para transportar requisições de documentos (BERNERS-LEE et al., 1994), o protocolo foi estendido ao longo dos anos para permitir uma troca assíncrona variada de dados entre o cliente e o servidor e, em alguns casos, até mesmo a construção de implementações específicas como *Representational State Transfer* (REST), muito utilizado para comunicação automática entre aplicativos (FIELDING, 2007).

Um conjunto de atividades para testes de segurança realizadas em um aplicativo web será representado por uma série de solicitações independentes de mensagens HTTP que deverão ser correlacionadas por meio de um mecanismo de persistência de estado de sessão do usuário.

2.2.4 Persistência do Estado de Sessão em Aplicativos Web

Apesar de utilizarem um protocolo de comunicação sem estado, caracterizado pela independência entre as mensagens e a não retenção de

informações do solicitante pelo servidor, os aplicativos web são essencialmente dependentes do estado de sessão do usuário.

É comum que um conjunto de atividades realizadas por um aplicativo web possua premissas e requisitos que, em muitas vezes, são representados por uma série de mensagens sequenciais HTTP executadas em ordem pré-estabelecida.

Utilize-se como exemplo um aplicativo web para compra de passagens aéreas. O usuário precisa transmitir uma mensagem ao aplicativo web contendo a origem e destino escolhidos para o voo e, a partir do resultado devolvido pelo servidor, escolher um voo que lhe seja mais adequado. Por final, o usuário irá transmitir seus dados financeiros para solicitar que o aplicativo faça a emissão do bilhete aéreo. Para executar com sucesso esta operação, todas as mensagens trocadas entre o cliente e o servidor devem respeitar uma sequência específica cuja responsabilidade de controle é exclusiva do aplicativo web.

Para permitir que este controle seja feito de maneira criteriosa em um protocolo sem estado, se faz necessária a utilização de um mecanismo de identificação de sessão baseado em uma sequência de dados com características de não repetição e imprevisibilidade, geralmente conhecida como *token*. Se a mensagem enviada pelo cliente for acompanhada de um *token*, o aplicativo web terá condições para recuperar informações pertinentes ao usuário e o estado de suas transações atuais.

Ao longo do tempo, os projetistas estenderam o protocolo para estabelecer mecanismos diversos para a transmissão deste identificador entre as mensagens do cliente para o servidor. São eles:

a) HTTP *Cookies*. Projetado pela Netscape Communication em 1994, trata-se de um mecanismo para envio de dados por meio do cabeçalho HTTP que permite o armazenamento persistente de dados do servidor para o cliente (navegador web). É amplamente utilizado para enviar o identificador (t*oken*) ao cliente que, por sua vez, passará a transmiti-lo em todas as mensagens subsequentes HTTP;

b) *Uniform Resource Information (URI) Tokens*. Trata-se de uma forma alternativa de anexar os identificadores de sessão no caminho solicitado (URI), como se fosse parte do pedido (ex: /recurso/emissao/1AB839CD);

c) Variáveis ocultas em formulários. Um mecanismo ainda bastante empregado em aplicativos mais simplificados. A aplicação web envia um formulário web que contém uma variável oculta com o identificador da sessão. Por exemplo, no caso de um formulário de acesso, o navegador web envia dados relativos ao usuário, senha e uma variável oculta contendo o *token* de controle de sessão.

2.2.5 Entradas de dados no protocolo HTTP

De acordo com Berners-Lee et al. (1994), URL são descritivos que representam o endereço de um recurso disponível na Internet. Em relação ao protocolo HTTP, Berners-Lee et al. descrevem um componente adicional para requisições aos aplicativos conhecido como *query string*, que permite que parâmetros de entrada de dados sejam enviados por meio de uma requisição do método *Get*. Adicionalmente, os autores também especificam o uso do cabeçalho do protocolo para transmissão de valores por meio de variáveis.

Também por meio do protocolo HTTP, especifica-se outro método para transportar parâmetros de entrada de dados a serem utilizados em aplicativos web. Trata-se do método *Post* que possui extensão para envio de um bloco de dados (FIELDING et al., 1999).

Desta forma, os seguintes métodos podem ser utilizados para a entrada de dados em aplicativos web:

a) Método *Get* por meio de *query string*;

b) Método *Post* por meio de seu bloco de dados;

c) Variáveis do cabeçalho HTTP em ambos os métodos.

2.3 OWASP

Fundada em setembro de 2001, a OWASP é uma organização sem fins lucrativos com o objetivo de promover e subsidiar projetos para o progresso da segurança em aplicativos web.

Apoiada por grandes empresas como IBM e HP, a organização também se dedica a publicação de padrões a serem utilizados pela indústria, como são os casos do *Application Security Verification Standard (*ASVS*)* e o OWASP Top 10, uma pesquisa das principais vulnerabilidades que afetam aplicativos web.

2.3.1 O *ranking* das principais vulnerabilidades dos aplicativos web

O OWASP mantém um projeto chamado de OWASP Top 10*,* uma publicação que contempla um *ranking* das dez principais vulnerabilidades que afetam as aplicações web. Trata-se de uma lista desenvolvida a partir de estatísticas providas pela indústria de tecnologia e organizações governamentais e se encontra na versão 2010.

A estrutura da lista apresenta as vulnerabilidades organizadas por nível de incidência e severidade de um eventual ataque, discutindo também os níveis de complexidade para exploração e detecção. São itens:

a) Falhas de Injeções de Código (A1), que são causadas pela falha na validação da entrada de dados fornecidos pelo usuário na aplicação web, permitindo a execução arbitrária de comandos operacionais e acesso não autorizado aos dados.

b) Falhas de *Cross-site Scripting* (A2), que ocorrem quando um parâmetro de entrada do usuário é processado diretamente pelo navegador web sem a validação adequada. O problema geralmente é explorado para permitir a injeção arbitrária de hipertexto (HTML) no documento produzido pelo aplicativo web.

c) Falhas de autenticação e Controle de Sessão (A3), que envolvem deficiências no processo de autenticação e controle de sessão do usuário, como por exemplo, permitir que usuários não autenticados executem ações privilegiadas no sistema.

d) Acesso direto inseguro a objetos (A4), que é uma deficiência em regular o acesso a recursos por meio de referências que não deveriam estar disponíveis sem o devido controle. Um bom exemplo são arquivos que deveriam estar protegidos pelo aplicativo, entretanto, podem ser acessados diretamente pelos usuários que conhecem o seu caminho.

e) *Cross-Site Request Forgery* (A5), que permitem que um usuário mal intencionado explore a sessão legítima de outro usuário (vítima) do sistema, utilizando mecanismos transparentes e de difícil detecção.

f) Falhas de configuração (A6), que são deficiências geralmente introduzidas na implementação e operação do aplicativo, causadas por algum erro de configuração. Um bom exemplo é a utilização de senhas previsíveis para acesso ao sistema.

g) Armazenamento Criptográfico Inseguro (A7), que trata da utilização de mecanismos ineficazes de criptografia para proteger dados sensíveis.

h) Falha de restrição de acesso a uma URL (A8), que permite que usuários burlem mecanismos de proteção para acessar uma determinada página do aplicativo.

i) Proteção Insuficiente da camada de Transporte (A9), que é uma deficiência no uso de tecnologia de proteção para o protocolo HTTP. Na maioria dos casos, está ligado com a ausência do uso de *Secure Socket Layers* (SSL) para o tráfego de informações sensíveis do aplicativo.

j) Falha de validação de redirecionamentos (A10). Uma vez que aplicativos web geralmente redirecionam os navegadores para outras localidades dentro da aplicação, esta falha permite que um usuário mal intencionado manipule esses redirecionamentos para páginas arbitrárias.

2.4 Os casos de uso e sua aplicabilidade na engenharia de software

De acordo com o conceito da engenharia de software, os casos de uso são uma descrição dos passos e ações a serem realizadas entre um usuário (ator) e um sistema de software para alcançar um determinado resultado (BITTNER; SPENCE, 2003). O usuário pode ser uma pessoa ou algo mais abstrato, como por exemplo, outro sistema de software ou processo manual.

Os casos de uso são amplamente utilizados na fase de concepção de um aplicativo para determinar suas características. Trata-se de uma forma de abordar especificação funcional que um sistema deve executar, sem a preocupação sobre como o software será implementado.

Geralmente os casos de uso são representados por descrições informais das atividades que serão executadas para alcançar um determinado resultado no sistema (COCKBURN, 2001). Pode ser elaborado em um parágrafo bem sumarizado de texto. Em algumas ocasiões, estas descrições podem ser ainda apoiadas com um diagrama que permita visualizar de maneira ilustrativa a sequência de atividades.

De acordo com Cockburn (2001), um bom caso de uso deve atender às seguintes prerrogativas:

a) Descrever o que o sistema deve fazer para que o ator alcance seu objetivo em especial, por exemplo, deve-se solicitar ao usuário o lugar de origem e destino para realizar uma consulta de disponibilidade de voo;

b) Não incluir nenhuma implementação em linguagem de programação;

c) Ter um nível apropriado de detalhes para a execução das atividades, por exemplo, o usuário deve entrar com uma senha válida para acessar o sistema de gestão de contas a pagar;

d) Não incluir detalhes sobre interfaces, telas ou qualquer detalhe técnico que possa inferir uma sugestão de como fazer.

2.5 A relação dos casos de uso e os testes de segurança

A partir do estudo realizado por Assad et al. (2010), constata-se que uma das formas eficazes de encontrar vulnerabilidades é entender em que parte da aplicação as falhas poderiam estar e, então, elaborar testes de segurança específicos para estes recursos.

De acordo com Cockburn (2001), entende-se que os casos de uso são uma fonte de documentação valiosa de descrição das atividades específicas funcionais de um aplicativo. Trata-se de uma receita – formal ou informal – de como executar uma série de tarefas para alcançar um determinado resultado no sistema. Além disso, por serem frequentemente utilizados, os casos de uso estariam amplamente disponíveis para serem consultados.

Desta forma, infere-se que os casos de uso podem ser utilizados para contribuir com o conhecimento detalhado das funcionalidades do sistema avaliado e subsidiar a elaboração de casos de testes de segurança mais eficazes nas aplicações web.

3 ESTADO DA ARTE DOS TESTES DE SEGURANÇA

Nesta seção se apresenta o estado da arte ao qual este trabalho se encontra inserido, incluindo uma descrição detalhada dos principais trabalhos que descrevem métodos de testes de segurança para detecção de vulnerabilidades.

Com o objetivo de destacar as principais diferenças, descrever-se-á um breve resumo dos trabalhos avaliados, destacando suas vantagens e desvantagens.

3.1 Análise dos principais métodos para testes de segurança

A análise das propostas mais recentes para os testes de segurança foi baseada em um estudo comparativo produzido por Shahriar e Zulkernine (2009), que levou em conta os principais trabalhos práticos para o tema.

Além disso, foi utilizado também como referência um estudo recente produzido por Bau et al. (2010) sobre o estado da arte do cenário dos testes de segurança caixa-preta, incluindo suas principais técnicas e ferramentas.

3.1.1 Método de testes de segurança caixa-preta

De acordo com Michael e Radosevich (2009), o método de testes de segurança de caixa-preta é uma técnica para avaliar um aplicativo web em tempo de execução, sem a necessidade de utilização do seu código-fonte.

A principal técnica empregada para a condução dos testes de segurança caixa-preta é o uso de bibliotecas de ataques previamente conhecidos para identificar um padrão positivo de vulnerabilidade na aplicação avaliada (ARKIN, 2005). O objetivo é executar os casos de teste às cegas, sob a mesma perspectiva de um usuário mal intencionado, ou seja, sem o conhecimento prévio da arquitetura e das funcionalidades da aplicação.

Sob o ponto de vista operacional, este método tem a vantagem de permitir que seja feita uma análise de segurança do ambiente de infraestrutura para produção do aplicativo web (MICHAEL; RADOSEVICH, 2009).

Adicionalmente, por averiguar uma grande base de ataques conhecidos, o teste tem bastante eficiência em detectar vulnerabilidades comumente exploradas em ambientes abertos como a Internet.

Para Bau et al. (2010), o método de execução dos testes de segurança caixa-preta consiste principalmente nas seguintes atividades:

a) O usuário deve selecionar os requisitos para iniciar os testes de segurança, incluindo a URL do aplicativo a ser avaliado, número de páginas web para navegação e as possíveis credenciais de acesso para autenticação no sistema;

b) Em seguida, o usuário deve iniciar a atividade de navegação e descoberta da aplicação, ou seja, acessar todas as páginas publicadas no aplicativo web e selecioná-las para testes;

c) Com a descoberta da aplicação, passa-se a realizar atividades de ataque contra as páginas web encontradas por meio dos seus pontos de entrada de dados, conforme descrito na seção 2.2.5;

Michael e Radosevich (2009) ainda estabelecem que as atividades dos testes de segurança caixa-preta são geralmente realizadas por meio de ferramentas caixa-preta que, por sua vez, se encarregam de executar as atividades de descobrimento do aplicativo (item b) e empregar as técnicas para os casos de teste (item c). Entre as principais técnicas de ataque para os casos de teste destacam-se:

- ***Fuzzing***, ou seja, a injeção de dados aleatórios ou sistematicamente gerados nos pontos de entrada de dados para identificar um comportamento inadequado;
- **Análise de dados** com o objetivo de identificar vazamento ou falta de proteção adequada para as informações;
- **Teste de sintaxe** para verificar comportamento inadequado por meio de valores ilegais enviados para os pontos de entrada de dados;

De acordo com a pesquisa realizada pela BSI-MM (2009), este é o principal método empregado para as atividades de teste de segurança no SDLC, contudo, possui algumas desvantagens conhecidas que podem limitar a eficácia para gestão de vulnerabilidades:

- Enquanto a utilização de uma biblioteca de ataques conhecidos tem eficácia em detectar problemas atuais, as falhas desconhecidas podem passar despercebidas, incluindo novos tipos de ataques ou aqueles que exploram especificidades do negócio (THOMPSON, 2003);
- Por se tratar de atividades de teste de segurança executados às cegas, ou seja, sem o conhecimento prévio, não é eficaz em detectar vulnerabilidades que possam existir na interação com as funcionalidades específicas do sistema (ASSAD et al., 2010);
- A maioria das ferramentas utilizadas para navegação não possui eficácia suficiente para levantamento de páginas web em aplicativos com conteúdo de alta interatividade, incluindo aqueles que exigem execução sistemática de *scripts* (BAU et al., 2010).

3.1.2 Método de testes de segurança caixa-branca

De acordo com McGraw e Potter (2004), os testes de segurança caixa-branca envolvem a avaliação da arquitetura da aplicação web por meio da análise do código-fonte. É geralmente muito eficiente para encontrar erros típicos de programação que podem introduzir vulnerabilidades nos aplicativos web, incluindo erros específicos oriundos de determinadas linguagens de programação.

O método de execução destes testes é realizado por meio da busca de padrões incomuns de programação em cada um dos arquivos que compõem o código-fonte do aplicativo web, incluindo uso de funções ou bibliotecas inseguras.

Por se tratar de uma análise intensa e extensa de varredura de arquivos, sua execução geralmente é apoiada por ferramentas caixa-branca que

automatizam a busca destes padrões em grandes volumes de código-fonte (MCGRAW; POTTER; 2004).

Apesar de ser uma atividade de teste relevante para o SDLC, deve ser tratada como técnica complementar para gestão de vulnerabilidades. Dentre as suas principais desvantagens, destacam-se dois pontos bastante comuns na execução dos testes:

- Por se tratar de uma análise baseada em busca de um conjunto de padrões positivos, os testes podem resultar em um grande volume de falsos-positivos, ou seja, possíveis eventos de erro que não configuram em uma vulnerabilidade de segurança (MCGRAW; POTTER, 2004);
- Além disso, a simples interpretação do código-fonte não permite que se infiram ou forneçam casos de utilização do aplicativo e, por conseguinte, expor funcionalidades críticas do sistema que deveriam ser priorizadas pelos testes de segurança;
- Por exigir acesso ao código-fonte completo da aplicação, essas análises podem não ser eficazes no caso de aplicativos web mais complexos que utilizam serviços ou componentes externos que não possuem código-fonte disponível (CANFORA; DI PENTA, 2006).

3.1.3 Método WAVES (HUANG et al., 2004)

O método WAVES é acrônimo para *Web Application Vulnerability and Error Scanner* e, de acordo com seus autores, trata-se de uma série de atividades para avaliar URL dos aplicativos web por meio de uma navegação controlada. As principais características do método WAVES incluem:

a) Para realizar a avaliação do aplicativo web não é necessário utilizar seu código-fonte como entrada;

b) Tal como os testes caixa-preta, utiliza-se um navegador web para identificar as páginas publicadas do aplicativo web por meio de sua interface pública e, a partir destas informações, realiza os casos de teste de segurança nos pontos de entrada de dados;

c) Baseia-se na detecção de classes de vulnerabilidades específicas tais como *Cross-site scripting* e injeção de SQL;

d) Implementa, por meio de uma ferramenta, a automatização da captura dos dados de navegação do aplicativo web.

O método WAVES, de acordo com estudo de Shahriar e Zulkernine (2009), é muito similar aos testes de segurança caixa-preta, navegando pela URL do aplicativo web para identificar os recursos a serem testados. Desta forma, compartilham algumas desvantagens, a saber:

- Por se tratar de um processo de descoberta de páginas web automatizado, não permite o fornecimento de interações específicas dentro do aplicativo, incluindo casos de uso que exponham funcionalidades importantes;
- De acordo com os autores, os resultados demonstraram problemas em processar e interpretar conteúdo dinâmico, por exemplo, *scripts*, restringindo de maneira extensiva a identificação de recursos em aplicativos web com alto nível de interatividade.
- O método foi construído com a premissa de avaliar duas classes de vulnerabilidades bem específicas, portanto, não restringindo a extensibilidade para outros tipos de detecção de falhas de segurança.

3.1.4 SecuBat (KALS et al., 2006)

SecuBat é um método proposto para a realização de testes de segurança por meio da navegação automática do aplicativo web. Da mesma maneira que o método WAVES proposto por Huang et al. (2004), o SecuBat captura as transações e passa a realizar casos de teste de segurança contra os pontos de entrada de dados das URL encontradas. De acordo com o artigo publicado, o SecuBat possui a seguinte arquitetura para execução dos testes de segurança:

a) **Componente de navegação**, para acesso às páginas do aplicativo web e captura automática dos pontos de entrada de dados por meio de formulários web;

b) **Componente de ataque**, que passa a realizar os ataques nos campos disponíveis nos formulários web;

c) **Componente de análise**, que analisa a resposta do servidor web para identificar possíveis falhas e vulnerabilidades.

Apesar de disponibilizar um amplo arcabouço para implementação de novas classes de ataque, o SecuBat possui algumas deficiências que podem dificultar a análise de aplicativos web mais dinâmicos e com alto grau de interatividade, a saber:

- Os casos de uso são montados a partir da leitura de formulários web disponíveis nas páginas encontradas no aplicativo. Enquanto esta técnica pode expor algumas das funcionalidades do aplicativo, pontos de entrada de dados não são necessariamente descritos apenas em formulários;
- Da mesma maneira que os testes caixa-preta e a proposta do WAVES, o método SecuBat prioriza a análise automática do aplicativo web, não permitindo a entrada de casos de uso específicos criados pelo usuário do aplicativo.

3.1.5 ARDILLA (KIEZUN et al., 2006)

O ARDILLA é uma proposta de um método híbrido para avaliar e detectar falhas de segurança em aplicativos web. Segundo os autores, testes caixa-preta e ferramentas que interagem somente com o tempo de execução não conseguem tirar vantagem do conhecimento do estado interno do aplicativo para construir casos de teste de segurança.

Como os métodos anteriores, O ARDILLA também foi desenvolvido para detectar *cross-site scripting* e injeção SQL por meio do seguinte fluxo de atividades para execução dos testes:

a) O usuário fornece como entrada o código-fonte do aplicativo web e seu banco de dados para testes;

b) A partir do código-fonte, o ARDILLA infere uma lista de parâmetros necessários para rodar cada uma das transações;

c) Em seguida, por meio de um componente de execução, interpreta e executa cada um dos códigos-fontes e identificar as regiões críticas que podem estar suscetíveis a falhas de segurança;

d) Por meio de um gerador de ataques, o ARDILLA utiliza as regiões críticas identificadas no item anterior e passa a modificar os parâmetros de entrada para simular um possível ataque. Pela monitoração do banco de dados, identifica se os ataques de injeção de SQL e *cross-site scripting* foram bem sucedidos;

Apesar de ser uma abordagem distinta, potencializada pelo amplo conhecimento das funcionalidades por meio do código-fonte e da interface em tempo de execução, o ARDILLA possui inúmeras restrições para realizar testes práticos de segurança no SDLC:

- O ARDILLA requer acesso direto ao código-fonte e o utiliza como referência para especificar transações. Segundo Kals et al. (2006), os ambientes cada vez mais heterogêneos de desenvolvimento e a complexidade dos aplicativos web são um desafio cada vez maior para a análise do código-fonte. Para aplicações mais interativas, as transações do aplicativo podem ser compostas de outros serviços cujo código-fonte não está disponível (CANFORA; DI PENTA, 2006);
- Por tratar de processamento e interpretação de código-fonte, a proposta se limita a um tipo de plataforma e linguagem de programação, no caso, PHP;

- Da mesma maneira que os outros testes anteriores, o ARDILLA limita bastante o escopo da análise; a construção dos casos de uso é realizada de maneira automática, não permitindo que uma série de passos seja seguida para se alcançar uma determinada funcionalidade.
- Por final, o método ARDILLA foi baseado em duas classes específicas de vulnerabilidade, restringindo sua extensibilidade para outros tipos de falhas de segurança.

3.1.6 D-WAV (ZHANG et al., 2010)

O método D-WAV, acrônimo para *Detector of Web Application Vulnerabilities,* é um conjunto de técnicas para detecção de vulnerabilidades cujo principal requisito é a entrada de documentos hipertexto (HTML) contendo formulários web que representem os pontos de entrada de dados do sistema.

Da mesma forma que o SecuBat (KALS et al., 2006), o método D-WAV possui uma série de desvantagens para os testes de segurança em aplicativos com alto nível de interatividade:

a) Devem-se submeter manualmente todos os formulários existentes no sistema para que o método seja eficaz, dificultando o processo de automatização e induzindo a falha humana;

b) Não permite o fornecimento de dados específicos que podem ser utilizados para construção de casos de uso especiais;

c) Não possui controle de sessão do usuário, portanto, ineficaz para testar aplicações que dependem deste mecanismo, tal como descrito na seção 2.2.4.

3.2 Conclusão do capítulo 3

Esta seção tratou de descrever os principais trabalhos que compõem o estado da arte dos testes de segurança para aplicativos web, incluindo suas

vantagens e desvantagens quando são analisados modelos de segurança sob a perspectiva do SDLC.

Inicialmente, foi analisado o teste de segurança caixa-preta que, de acordo com pesquisas conduzidas pelo BSI-MM (2009), trata-se de um dos principais métodos empregados na indústria para gestão de vulnerabilidades. Compreende uma série de técnicas que emprega um banco de dados de falhas conhecidas contra um aplicativo web, sem conhecimento prévio de suas funcionalidades. Verificou-se que sua eficácia é restrita aos problemas já conhecidos e que não envolvam funcionalidades específicas da aplicação web.

Adicionalmente, foi analisado o teste de segurança caixa-branca, que também se trata de um método tradicional com o objetivo de avaliar padrões de vulnerabilidade no código-fonte do aplicativo web. Desta forma, apesar de identificar erros comuns introduzidos pelo estilo ou linguagem de programação, trata-se de um método limitado à disponibilidade do código-fonte e a não dependência de nenhum componente externo em tempo de execução. Outra proposta também estudada, o método ARDILLA (KIEZUN et al.,2006), sofre das mesmas restrições, uma vez que necessita de acesso ao código-fonte e, como agravante, está limitado a uma única plataforma de desenvolvimento, o PHP.

Em seguida, tratou-se de analisar o método WAVES (Huang et al., 2004), que especifica uma série de atividades automáticas para descobrir recursos, construir casos de teste e testar a segurança de aplicativos web. Apesar de permitir que testes sejam automatizados, eles não são realizados com a premissa do conhecimento prévio do aplicativo web e, portanto, sofrem das mesmas limitações dos testes caixa-preta. Com pequenas vantagens operacionais como extensibilidade das classes de vulnerabilidade, outros métodos como SecuBat (KALS et al., 2006) e D-WAV (ZHANG et al., 2010) também sofrem as mesmas restrições.

Portanto, é possível identificar entre os métodos estudados a característica comum de existir restrições para simulação de um fluxo de atividades que represente uma funcionalidade importante do sistema. Desta forma, nos casos em que o sistema, por exemplo, requer que um usuário forneça uma credencial de acesso para permitir interação com suas

funcionalidades, o teste de segurança estaria restrito às áreas que não exigem autenticação e, portanto torna-se ineficaz para detectar vulnerabilidades em áreas importantes do aplicativo.

A falta de aderência dos testes ao fluxo de atividades funcionais da aplicação é apontada como um das principais causas da baixa qualidade dos testes de segurança (ASSAD et al., 2010). Em todos os métodos estudados, os requisitos para a execução dos testes de segurança são mínimos, geralmente sem a necessidade de conhecimento prévio do aplicativo, conforme estabelece Arkin et al. (2005), quando descreve as fragilidades dos testes de invasão.

As características positivas que permitem a extensão das classes de vulnerabilidades analisadas nos métodos de testes de caixa-preta e SecuBat (KALS et al., 2006) são desejáveis em um teste de segurança, entretanto, carecem da flexibilidade necessária para permitir a configuração de um fluxo de utilização específico e garantir o controle de sessão de usuário. Trata-se de característica imprescindível para acompanhar o nível de segurança durante todo o ciclo de vida do aplicativo.

Desta forma, conclui-se a existência de uma oportunidade para aproveitamento do conhecimento preexistente das interfaces funcionais de um aplicativo web, conforme descrito na seção 2.4, e detectar classes de vulnerabilidades em funcionalidades da aplicação web, o que não seria possível se fosse empregadas propostas tradicionais de testes.

Sob o ponto de vista comparativo, as seguintes características são vantajosas em relação aos trabalhos estudados, conforme estabelecido em resumo do quadro 1:

a) A independência da utilização do código-fonte do aplicativo, não restringindo a execução dos testes de segurança;

b) Capacidade para permitir que o teste seja automatizado para execução de maneira sistemática em todas as fases do SDLC, tanto para validar as correções aplicadas quanto para testar possíveis modificações das funcionalidades;

c) Amplo uso do conhecimento das interfaces funcionais do aplicativo, permitindo a configuração de casos de uso como entrada para navegação sem depender da descoberta automática de URL;

d) Extensibilidade para permitir que os testes de segurança não fiquem restritos a determinadas classes de vulnerabilidade

Por final, esta nova proposta tem a pretensão de ser uma complementação para as abordagens existentes. O objetivo é permitir o emprego conjunto das diferentes técnicas de testes de segurança em fases distintas de vida do aplicativo web, incluindo desenvolvimento, homologação e produção, permitindo, assim, uma mitigação mais eficiente de suas vulnerabilidades (GRAHAM et al. 2006).

Resumo do Estado da Arte dos testes de segurança para aplicativos web			
Nome do Método	Não precisa utilizar o código-fonte?	Permite utilizar conhecimentos funcionais do aplicativo?	Permite estender os testes para outras classes de vulnerabilidade?
Testes de segurança caixa-preta	Sim	Não	Sim
Testes de segurança caixa-branca	Não	Não	Sim
WAVES	Sim	Não	Não
SecuBat	Sim	Não	Sim
ARDILLA	Não	Sim	Não
D-WAV	Sim	Não	Sim
SCOUT	**Sim**	**Sim**	**Sim**

Quadro 1 – Resumo do Estado da Arte
Fonte: Elaborado pelo autor.

4 MÉTODO PARA TESTES DE SEGURANÇA EM APLICAÇÕES WEB

O objetivo desta seção é apresentar de maneira detalhada o método SCOUT utilizado para realização dos testes de segurança em aplicações web por meio do emprego de casos de uso.

Serão detalhados os requisitos e atividades necessários para o emprego dos casos de uso de um aplicativo web nos testes, incluindo os procedimentos para avaliação e detecção das classes de vulnerabilidade escolhidas.

4.1 Descrição geral do método

De acordo com a figura 1, o método SCOUT compreende uma série de atividades distintas que devem ser aplicadas sequencialmente, observando os requisitos que serão descritos detalhadamente nas próximas seções.

De maneira geral, o método pode ser dividido em três grandes fases, incluindo planejamento, execução e resultados. Em cada uma destas fases serão executadas atividades distintas cujos resultados deverão ser utilizados nas atividades posteriores.

Na fase de Planejamento, as atividades incluem:

- Atividade 1: Levantar e validar os requisitos necessários para os testes;
- Atividade 2: Planejar a execução dos casos de uso primários e secundários;

Na fase de Execução, estão incluídas as atividades:

- Atividade 3: Executar os casos de uso primários e secundários para capturar as transações relevantes;
- Atividade 4: Preparar os casos de testes para execução de acordo com as URL capturadas;
- Atividade 5: Executar os casos de testes.

Na fase de Resultados, a atividade será:

- Atividade 6: Analisar os resultados e gerar relatório.

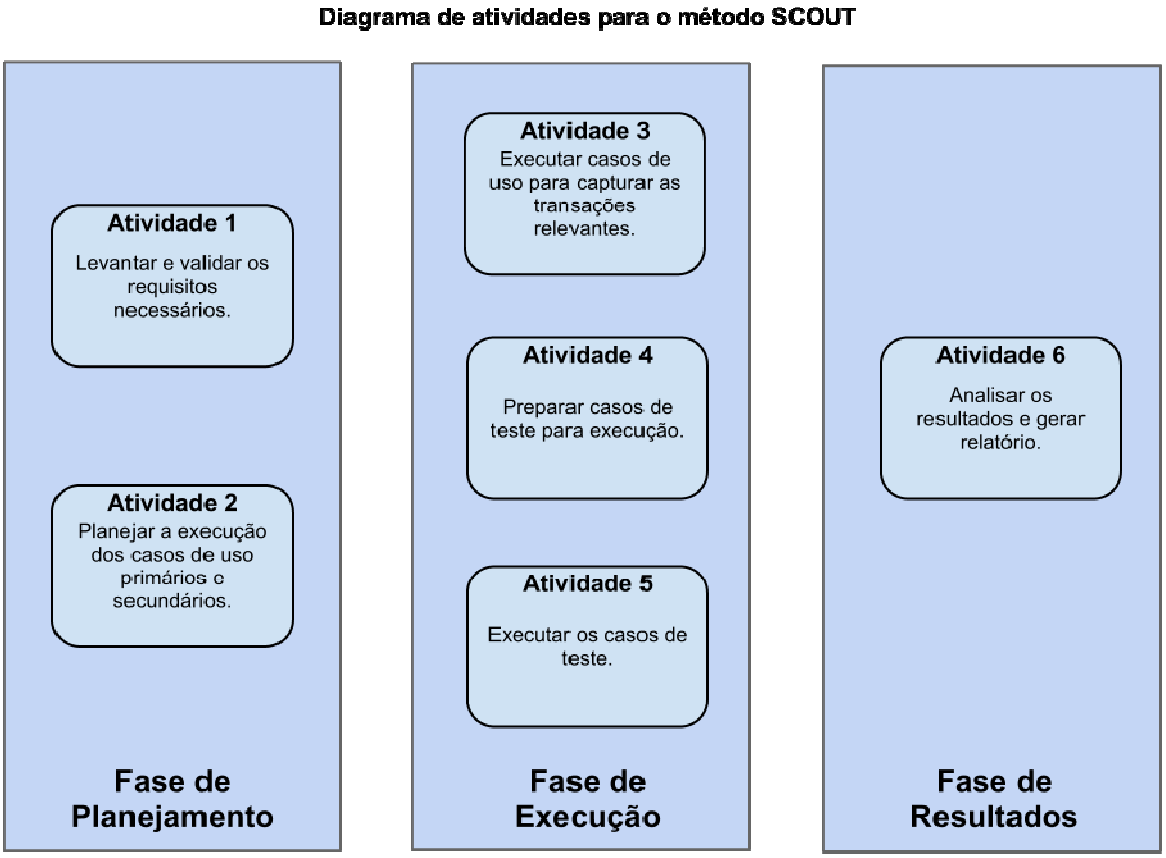

Figura 1 – Diagrama das atividades do método SCOUT
Fonte: Elaborado pelo autor.

4.2 Classificações especiais de casos de uso utilizadas no SCOUT

O método empregará duas classificações especiais de casos de uso para as atividades nas fases de planejamento, execução e resultados; a saber:

a) **Casos de uso primários:** são os casos de uso obrigatórios para a execução de quaisquer outros passos. São pré-requisitos para outros casos de uso, por exemplo, o processo de autenticar-se utilizando uma credencial válida em um sistema;

b) **Casos de uso secundários:** são todos os outros casos de uso do aplicativo que dependem da execução prévia de um caso de uso

primário. Por exemplo, a consulta do extrato de consumo de recursos em um aplicativo financeiro.

4.3 Detalhamento das fases do método SCOUT

A seguir são descritas de maneira detalhada as fases e as respectivas atividades a serem realizadas para a execução completa do método para os testes de segurança.

4.3.1 Fase de Planejamento

4.3.1.1 Atividade 1: Levantar e validar os requisitos necessários

Esta atividade é descrita em termos de requisitos que serão utilizados ao longo da execução do método SCOUT. O objetivo é levantar e validar os requisitos, conforme os critérios abaixo:

- **Requisito 1: Endereço do aplicativo web a ser avaliado:**

Trata-se do endereço utilizado pelo aplicativo web no formato de URL. Caso o aplicativo contenha mais de um endereço, estes também deverão estar descritos para permitir que a análise seja feita em todas as URL relevantes para o aplicativo.

- **Requisito 2: Descrição dos casos de uso primários e secundários.**

Esta é a lista de todos os casos de uso que serão empregados para a análise das classes de vulnerabilidades no aplicativo web. Os casos de uso primários serão utilizados para obter os valores de *tokens* válidos de controle de sessão do usuário, conforme descrito na seção 2.2.4. Os casos de uso secundários consistem nos passos que serão realizados dentro do sistema a partir do uso dos *tokens* obtidos anteriormente.

- **Requisito 3: Definição da sequência de execução dos casos de uso**.

A partir do levantamento de todos os casos de uso primários e secundários especificados no requisito 2, a atividade 1 exige que a sequência de execução dos casos de uso do aplicativo web esteja definida.

Por exemplo, um caso de uso primário de autenticação no aplicativo web deve preceder o caso de uso secundário para busca de disponibilidade de veículos para locação. Este, por sua vez, deve preceder o caso de uso secundário para solicitação de reserva de veículo.

- **Requisito 4: Definição da classe de vulnerabilidade para testes**

Os testes de segurança serão realizados com base em uma ou mais classes de vulnerabilidades utilizadas para preparação e construção dos casos de teste.

O levantamento dos requisitos deve ser organizado de forma a permitir que se faça uma verificação detalhada antes do início dos testes. O resultado desta atividade será utilizado como entrada para as atividades 2 da fase de planejamento; atividades 3, 4 e 5 da fase de execução; e atividade 6 da fase de resultados.

4.3.1.2 Atividade 2: Planejar a execução dos casos de uso primários e secundários

Esta atividade compreende o planejamento da execução dos casos de uso primários e secundários de acordo com os requisitos para utilização de mecanismos de identificação e controle de sessão do usuário (*tokens)* no aplicativo web*.* Será utilizado como entrada a definição dos casos de uso primários e secundários, especificado pelo requisito 2 da atividade 1.

De acordo com a seção 2.4, os casos de uso são a descrição de uma série de tarefas a ser executada de maneira pré-estabelecida para se alcançar uma determinada funcionalidade do sistema. Entretanto, conforme os conceitos introduzidos na seção 2.2.4, a execução de uma série de passos correlacionados em um aplicativo web pressupõe a utilização de um mecanismo de identificação e controle de sessão que seja persistente ao longo de toda sua avaliação.

Esta atividade de planejamento deverá, por meio da avaliação das transações dos casos de uso primários, identificar quais os *tokens* que serão utilizados com base nas seguintes referências:

- **HTTP *cookies:*** São os valores transmitidos no cabeçalho HTTP por meio do identificador *Set-Cookie*;
- **URI *tokens*:** São valores aleatórios utilizados para compor os caminhos URI;
- **Variáveis ocultas em formulários:** São as variáveis do tipo *hidden* incluídas nos formulários web.

Uma vez identificados os valores com base nos itens acima, deve-se empregar os seguintes critérios para verificação e coleta dos *tokens* válidos do sistema:

a) **Imprevisibilidade.** *Token* deve ser uma sequência de dados aleatórios que não devem ser facilmente previsíveis no contexto do aplicativo.

b) **Valores únicos ao longo do tempo.** Por se tratar de um dado gerado por meio de algoritmos aleatórios, o *token* é um valor único e distinto entre diferentes sessões de usuário ao longo do tempo;

c) **Constante durante a sessão.** Os valores de um *token* devem permanecer constantes ao longo de todas as atividades dentro da mesma sessão de usuário.

d) **Variáveis ao longo de múltiplas sessões.** Em razão das características descritas nos itens a e b, os valores de um *token* são sempre diferentes se comparados entre sessões distintas.

Os valores coletados que atenderem todos os critérios deverão ser armazenados para uso posterior na atividade 3 da fase de Execução.

4.3.2 Fase de Execução

4.3.2.1 Atividade 3: Executar os casos de uso primários e secundários para capturar as transações relevantes

Esta atividade compreende a execução dos casos de uso primários e secundários para a coleta das transações relevantes utilizadas no aplicativo web. Serão utilizados como entrada desta atividade os requisitos 1, 2 e 3 da atividade 1.

Por se tratar de uma série de passos pré-estabelecidos dentro de um sistema para alcançar uma determinada funcionalidade, o fluxo de execução dos casos de uso pode ser descrito como uma sequência de URLs capturadas por meio da monitoração das transações envolvidas em cada um dos casos. Estas URLs, por sua vez, serão utilizadas para a preparação dos casos de teste na atividade 4.

De acordo com os requisitos do aplicativo web, os casos de uso geralmente possuem uma sequência pré-estabelecida para execução, portanto, deve-se utilizar como entrada o fluxo descrito pela especificação do requisito 3 da atividade 1.

De maneira geral, esta atividade deve ser executada por meio do algoritmo listado a seguir:

1. Obter o primeiro caso de uso por meio da sequência especificada pelo requisito 4 da atividade 1;
2. Obter os passos a serem executados no caso de uso por meio da lista especificada no requisito 2 da atividade 1;
3. Para cada passo descrito no item 2, obter a URL da transação invocada;
4. Verificar se a URL pertence a um dos endereços descritos no requisito 1 da atividade 1. Em caso positivo, subscrever a URL a uma lista de controle. Caso contrário, desprezar a informação;
5. Verificar se os valores dos *tokens* de controle válido da sessão continuam constantes. Caso tenham sido alterados, executar

novamente a atividade 3 do SCOUT e, em seguida, continuar no item 6;

6. Caso exista um novo caso de uso na sequência da lista especificada no requisito 3 da atividade 1, repetir o item 2. Caso contrário, concluir a execução.

Por meio das iterações do item 4 deste algoritmo acima, será produzida uma lista de URL que deverá ser utilizada como entrada para a atividade 4.

4.3.2.2 Atividade 4: Preparar os casos de testes para execução

Esta atividade trata da preparação da lista de URLs produzida na atividade 3 para construção dos casos de teste de detecção das classes de vulnerabilidades especificadas no requisito 5 da atividade 1.

Conforme estabelecido pela seção 2.2.5, as URLs do aplicativo web possuem três pontos distintos de entrada de dados, o método *Get*, *Post* e o cabeçalho do protocolo HTTP. A construção dos casos de teste consiste na implantação prática dos padrões de ataque da classe de vulnerabilidade testada para estes dois métodos. Em cada uma das URLs utilizadas, correlaciona-se um comportamento típico esperado da resposta do aplicativo para configurar a existência da vulnerabilidade.

Por exemplo, para verificar se uma URL é vulnerável a problemas de ataque de transbordamento de *buffer (buffer overflow)*, deve-se trocar o valor das variáveis de entrada por uma sequência de dados aleatória e contínua de 500 ou mais *bytes*. Para determinar se a vulnerabilidade existe, espera-se que o servidor retorne com um erro interno de processamento com o código 500.

Como resultado desta atividade, será produzida uma lista de URLs que implantem as classes de ataque a serem testadas, incluindo o comportamento esperado para cada resposta do servidor. A combinação destas informações será o caso de teste.

Cada classe de vulnerabilidade pressupõe um caso de teste distinto e a lista dos casos produzidos nesta atividade será utilizada como entrada para a atividade 5.

4.3.2.3 Atividade 5: Executar os casos de testes

Esta atividade compreende a execução da lista de casos de teste produzida na atividade anterior. Seu fluxo de execução deverá seguir o algoritmo listado abaixo:

1. Selecionar o primeiro caso de teste disponível a partir da lista de casos de teste produzida pela atividade 5;
2. A partir do caso de teste selecionado, obter a primeira URL;
3. Invocar a URL por meio do protocolo HTTP e coletar o documento HTML de resposta do servidor web. Armazenar o pedido e resposta HTTP; e o documento HTML em uma lista separada de resultados;
4. Verificar se os *tokens* de controle de sessão da atividade 3 continuam constantes e sem alteração. Em caso negativo, executar novamente a atividade 3 e seguir no item 5;
5. Caso exista mais URL disponíveis no caso de teste, obter a próxima URL e seguir para o item 3. Caso contrário, seguir para o item 6;
6. Caso existam mais casos de teste disponíveis na lista de casos de teste produzida pela atividade 5, selecionar o caso e seguir no item 2. Caso contrário, concluir a atividade.

Esta atividade irá produzir uma lista de resultados contendo os pedidos e respostas HTTP, incluindo o documento HTML, da execução dos casos de teste para serem utilizados na fase de Resultados.

4.3.3 Fase de Resultados

4.3.3.1 Atividade 6: Analisar os resultados e gerar relatório

Nesta atividade será realizada a análise dos resultados produzidos na atividade 5 para detecção de vulnerabilidades e a geração do relatório dos testes de segurança.

Para execução desta atividade será utilizado um algoritmo que, por sua vez, terá como entrada a lista de resultados da atividade 5. Segue a especificação do algoritmo para análise de resultados:

1. A partir da lista de resultados gerada na atividade 5, obter a primeira URL disponível.
2. A partir da URL, obter os resultados do pedido e resposta HTTP, incluindo o documento HTML. Comparar estes dados com o padrão de comportamento positivo esperado do aplicativo web e definido na atividade 4.
3. Em caso positivo, classificar a URL como vulnerável e subscrevê-lano relatório do teste de segurança.
4. Verificar a existência de uma próxima URL disponível a partir da lista de resultados da atividade 5. Em caso positivo, seguir para o item 2. Caso negativo, concluir a atividade.

Os seguintes critérios deverão ser observados na produção do documento final:

a) Inclusão da URL e dos detalhes relativos ao pedido e resposta HTTP, bem como o documento HTML enviado pelo servidor;
b) Descrição sumarizada do padrão positivo encontrado no comportamento do aplicativo para detecção da vulnerabilidade;
c) Incluir, caso estejam disponíveis, referências técnicas que descrevam a classe de vulnerabilidade e os padrões de ataque.

Esta atividade irá concluir a execução do método SCOUT, produzindo um relatório descritivo das falhas de segurança encontradas no aplicativo web baseado nas classes de vulnerabilidade escolhidas no requisito 5 da atividade 1.

4.4 Fluxo de execução do método SCOUT

De acordo com o fluxo descrito de maneira ilustrativa na figura 2, a execução do SCOUT inicia-se por meio da atividade 1 da fase de Planejamento, quando se deve levantar e validar os requisitos necessários para emprego ao longo de todo o teste de segurança. Em seguida, a atividade 2 recebe como entrada os requisitos produzidos pela atividade 1 e deve planejar a execução dos casos de uso, avaliando os possíveis mecanismos de identificação do controle de sessão que deverão ser utilizados durante os testes.

Na fase de Execução, a atividade 3 executa os casos de uso primários e secundários utilizando como entrada os requisitos da atividade 1 e os *tokens* da atividade 2, produzindo, como resultado, uma lista de URL da aplicação web. A atividade 4 utiliza a lista da atividade 3 para preparar uma nova lista com os casos de teste de segurança. Em seguida, a lista de casos de teste da atividade 4 será utilizada para a execução propriamente dita dos casos de teste na atividade 5. Como resultado da atividade 5, será produzida uma lista de resultados que representa a análise dos casos de teste de segurança. Por final, na fase de Resultados, a atividade 6 irá utilizar a lista de resultados produzida na atividade 5 para análise e geração de relatórios de detecção de vulnerabilidades do aplicativo web.

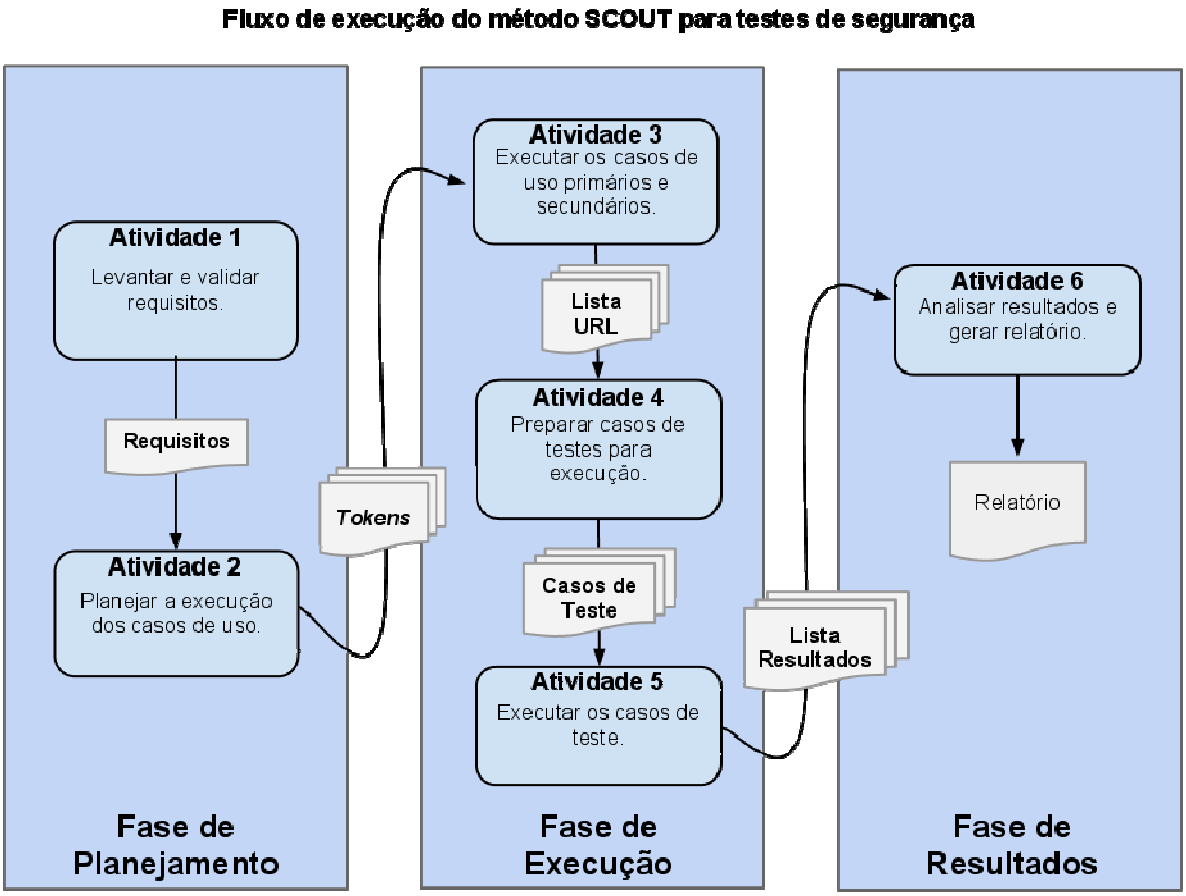

Figura 2 – Fluxo de execução das atividades do método SCOUT
Fonte: Elaborado pelo autor.

4.5 Conclusão do capítulo 4

O método SCOUT compreende seis atividades distintas distribuídas em três fases, incluindo: Planejamento, com o levantamento de requisitos e planejamento da execução dos casos de uso primários e secundários; Execução, com a execução dos casos de uso primários e secundários, preparação dos casos de teste e execução dos casos de teste; e, por final, Resultados, incluindo a análise dos resultados e geração de relatório.

A execução do método é realizada por meio da entrada de quatro requisitos básicos, incluindo:

1. Endereço do aplicativo web a ser avaliado;
2. Descrição dos casos de uso primários e secundários;

3. Definição da sequência de execução dos casos de uso;
4. Definição da classe de vulnerabilidades para testes.

Quanto aos casos de uso utilizados como requisitos de entrada, deve-se classificá-los como casos de uso primários, aqueles obrigatórios para execução de outras transações ou casos de uso; e os casos de uso secundários, todos os outros casos de uso que dependem de execução dos casos de uso primários.

O fluxo de execução será iniciado por meio da atividade 1, com o levantamento dos requisitos básicos, e realizado de maneira sequencial, observando todos os requisitos de entrada e saída, até a análise dos resultados e produção do relatório na atividade 6.

Conclui-se que o método SCOUT tem por principal característica estabelecer, de maneira eficiente e eficaz, a utilização do conhecimento de funcionalidades específicas do sistema, por meio dos casos de uso, para execução dos testes de segurança. Destacam-se como suas principais premissas:

a) Dar flexibilidade necessária para a configuração de casos de uso, tanto os primários ou secundários, permitindo a interação controlada com transações específicas do aplicativo, por exemplo, aquelas que necessitem do gerenciamento do controle de sessão de usuário;
b) Propiciar um método que permita a adaptação dos casos de teste a qualquer classe de vulnerabilidade desejada;
c) Permitir que os testes sejam repetidos de maneira sistemática e possam ser utilizados para certificar as correções ou modificações realizadas em fases distintas do SDLC.

O próximo capítulo irá tratar da validação do SCOUT, a ser realizada por meio de um ambiente experimental de testes com o objetivo de produzir resultados que demonstrem a eficácia do SCOUT em face à sua comparação com os outros métodos estudados.

5 VALIDAÇÃO DO TRABALHO

Esta seção irá descrever os detalhes sobre o ambiente dos testes, incluindo plataformas, ferramentas e critérios a serem observados. Para garantir a imparcialidade dos testes, será utilizado um ambiente comum para todas as execuções, conservando as premissas semelhantes para cada um dos métodos empregados.

5.1 Trabalhos utilizados para comparação

Este trabalho será validado por meio de testes experimentais de prova de conceito com o objetivo de comparar a eficácia do SCOUT em relação aos seguintes métodos descritos no estado da arte:

a) Teste de segurança caixa-preta (BAU et al., 2010);

b) SecuBat (KALS et al, 2006);

c) WAVES (HUANG et al., 2004);

d) D-WAV (ZHANG et al., 2010).

Estes métodos foram escolhidos com base na pesquisa de estado da arte realizada pelo artigo de Shahriar e Zulkernine (2009), por tratar de atividades específicas com o foco em testes de segurança para aplicativos web e sem a necessidade da utilização do código-fonte. Os métodos a e b foram testados por meio de ferramentas disponibilizadas pelos autores. Os demais métodos c e d não disponibilizaram ferramenta de apoio para implementação, portanto, resultados só podem ser inferidos com base em execução teórica.

5.2 Critérios de comparação entre os trabalhos

Para comparação entre o SCOUT e os métodos escolhidos, foi utilizado o critério de assertividade, ou seja, uma análise qualitativa do número de vulnerabilidades encontradas e não encontradas para determinar o nível de detecção de cada um dos métodos. Utilizou-se como referência a lista de

falhas catalogadas pelo banco de dados de vulnerabilidades do *National Vulnerability Database* (NVD) do NIST.

5.3 Ambiente para prova de conceito

A prova de conceito foi realizada em ambiente experimental controlado e independente de fatores externos, incluindo, por exemplo, uma rede privativa para comunicação entre os componentes.

Para que os testes fossem realizados de maneira representativa, utilizou-se um aplicativo web amplamente disponível e utilizado na Internet.

5.3.1 Aplicativo web para os testes de segurança

A escolha do aplicativo web para realização dos testes de prova de conceito baseou-se nos seguintes critérios:

a) Aplicativo simplificado e com ampla base de uso e distribuição;

b) Licença flexível para utilização em laboratório;

c) Histórico conhecido de vulnerabilidades para testes;

d) Ampla documentação para instalação.

Foram analisadas diferentes ferramentas e aquela que melhor preencheu os requisitos foi o aplicativo *wordpress*, cuja finalidade é implementar uma plataforma para publicação de *blogs*. De acordo com as estatísticas de seu sítio, trata-se de um aplicativo amplamente difundido na Internet, sendo utilizado por mais de um milhão de endereços individuais. A ferramenta é a plataforma de comunicação de grandes empresas multinacionais, incluindo Sony, General Motors, UPS e Volkswagen.

Além disso, de acordo com dados do NVD (NIST, 2011), o aplicativo *wordpress* possui um extenso histórico de mais de 300 vulnerabilidades reportadas desde 2004 (veja figura 3).

Outros aplicativos exclusivos para testes de segurança, tais como *HackMe* e o *WebGoat*, foram considerados na análise, entretanto, o fluxo de execução não depende da utilização de casos de uso e, portanto, não refletiriam de maneira adequada a análise dos objetivos deste trabalho.

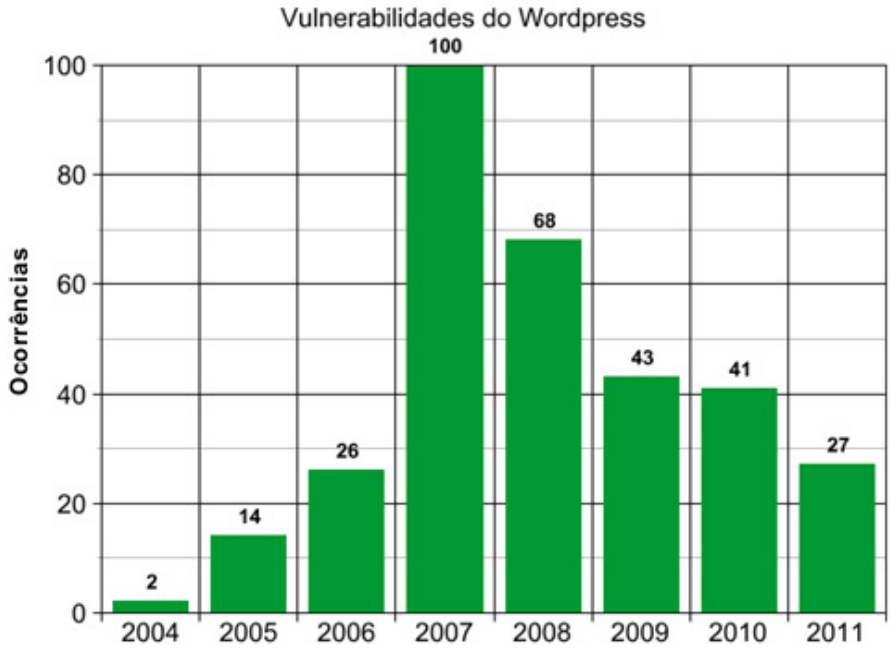

Figura 3 – Falhas de segurança encontradas no *wordpress* desde 2004.
Fonte: *National Vulnerability Database* – NIST (2011).

De acordo com sua documentação oficial, o *wordpress* possui os seguintes requisitos para instalação:

a) Módulo de tempo de execução do código PHP;

b) Servidor web gratuito Apache;

c) Banco de dados gratuito MySQL;

d) Computador para hospedar esses serviços que utilize um sistema operacional com ampla capacidade de comunicação, sendo os mais recomendados o Windows ou Linux.

5.3.2 Topologia do ambiente

A topologia do ambiente está definida de acordo com a figura 4 e a instalação será guiada por meio de documentação disponível nos sítios oficiais dos fabricantes ou distribuidores.

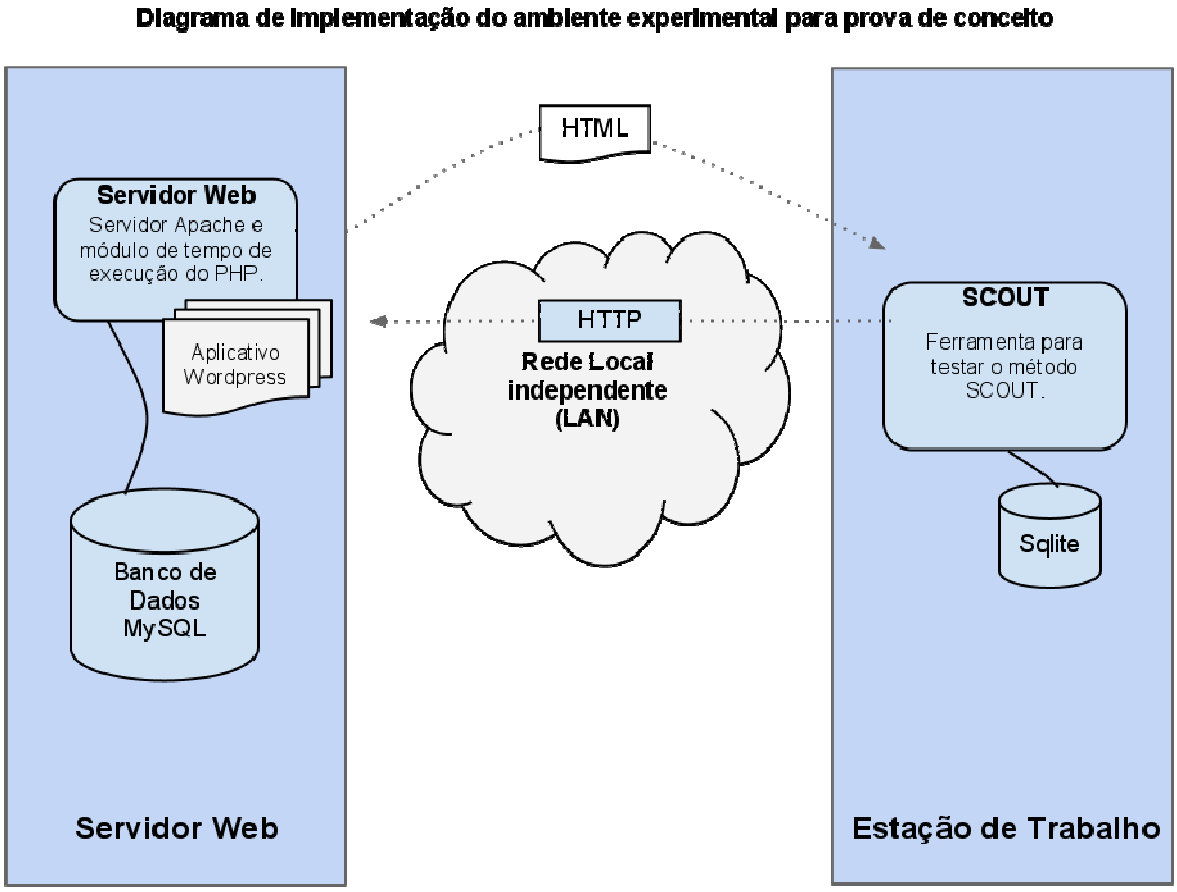

Figura 4 – Topologia do ambiente de testes de prova de conceito do SCOUT
Fonte: Elaborado pelo autor.

Os componentes do ambiente de teste e suas respectivas configurações de *hardware* e *software* são:

- **Servidor web:** Servidor com processamento de arquitetura x86, 32-bits, 1 GB de memória RAM e 10 GB de disco rígido. Rodando sistema operacional Linux com servidor web Apache, plataforma PHP e banco de dados MySQL. Todos os aplicativos são gratuitos e com código-fonte abertos;

- **Estação de trabalho:** Máquina com processamento de arquitetura x86, 32-bits, 1 GB de memória RAM e 10 GB de disco rígido. Rodando sistema operacional Windows. Os componentes para a ferramenta de testes serão descritos posteriormente nesta seção.

5.4 Casos de uso para a aplicação

Por meio de análise da documentação do aplicativo *wordpress*, foi possível extrair um conjunto de casos típicos de utilização relevante para os testes de prova de conceito.

Por se tratar de um aplicativo de publicação de conteúdo escolheu-se realizar os testes a partir de um perfil de acesso válido e com privilégios mínimos de leitura e escrita no sistema. Os casos de uso escolhidos são descritos detalhadamente nas próximas seções.

5.4.1 Caso de uso primário 1: Autenticação do usuário

Trata-se de um caso de uso primário que tem por objetivo autenticar e identificar um usuário válido no sistema.

Seus passos incluem:

1. Acessar URL principal do *blog;*
2. Acessar o *link* de acesso à área de autenticação;
3. Submeter dados de usuário teste e senha teste no formulário de autenticação;
4. Acessar o *link* painel de controle.

5.4.2 Caso de uso secundário 2: Listagem de artigos

Este caso de uso será utilizado para testar funcionalidades de leitura da plataforma de publicação de conteúdo, permitindo que usuário válido do sistema realize uma consulta básica dos últimos artigos publicados.

Seus passos incluem:

1. Acessar o endereço principal do sítio de testes;
2. No Menu da direita, escolher os artigos de categoria geral (“Uncategorized”);
3. No primeiro artigo publicado, clicar no título principal para visualizar seu conteúdo integral;
4. Mover-se para o próximo artigo clicando em seu título no canto superior esquerdo.

5.4.3 Caso de uso secundário 3: Publicação de artigo

Este caso de uso será utilizado para testar funcionalidades de escrita da plataforma de publicação de conteúdo, permitindo que um usuário válido do sistema faça a publicação de um novo artigo.

Seus passos incluem:

1. Acessar o *link* do painel de controle*;*
2. No menu da esquerda, acessar o *link* “novo artigo”;
3. No formulário de inclusão de novo artigo, entrar com título e conteúdo genérico. Utilizar gerador de conteúdo genérico Lero-Lero;
4. Acessar botão concluir para terminar a inclusão.

5.4.4 Caso de uso secundário 4: Publicação de comentário

Este caso de uso será utilizado para testar funcionalidades de escrita da área pública do sítio web. O objetivo é simular a entrada de um comentário para o novo artigo criado no caso de uso anterior.

Seus passos incluem:

1. Acessar o endereço principal do *blog;*
2. Na lista de artigos, selecionar o último artigo incluído pelo caso de uso anterior;

3. Ir até o rodapé do artigo e localizar formulário para entrada de comentário;
4. No formulário de entrada de comentário, entrar com endereço eletrônico e texto de comentário genérico. Utilizar gerador de conteúdo genérico, por exemplo, textos em latim capturados na Internet.

5.4.5 Caso de uso secundário 5: Remoção de comentário

Este caso de uso será utilizado para testar funcionalidades do painel administrativo do sítio web. O objetivo é efetivar a remoção do comentário criado no caso de uso anterior.

Seus passos incluem:

1. Acessar a área administrativa do *blog;*
2. No painel principal, entrar na seção de gerenciamento de comentários;
3. Localizar o comentário inserido no caso de uso anterior;
4. Clicar na ação de “Remover de comentário”.

5.5 Ferramenta para prova de conceito

5.5.1 Método SCOUT

Para testar o conceito proposto pelo SCOUT foi desenvolvida uma ferramenta para implantação, automatização e acompanhamento de suas atividades.

A plataforma utilizada para construção da ferramenta foi o ambiente operacional Windows 32-bits, uma vez que dispõe de ferramentas de desenvolvimento rápido e integração transparente para os seguintes componentes necessários para realização das atividades:

- SQLite3, pequeno banco de dados para ambientes embarcados baseado na linguagem SQL;
- Engenho de comunicação HTTP integrado a um navegador web de mercado (Internet Explorer);
- Interpretador de linguagem de scripts Lua para construção de casos de teste.

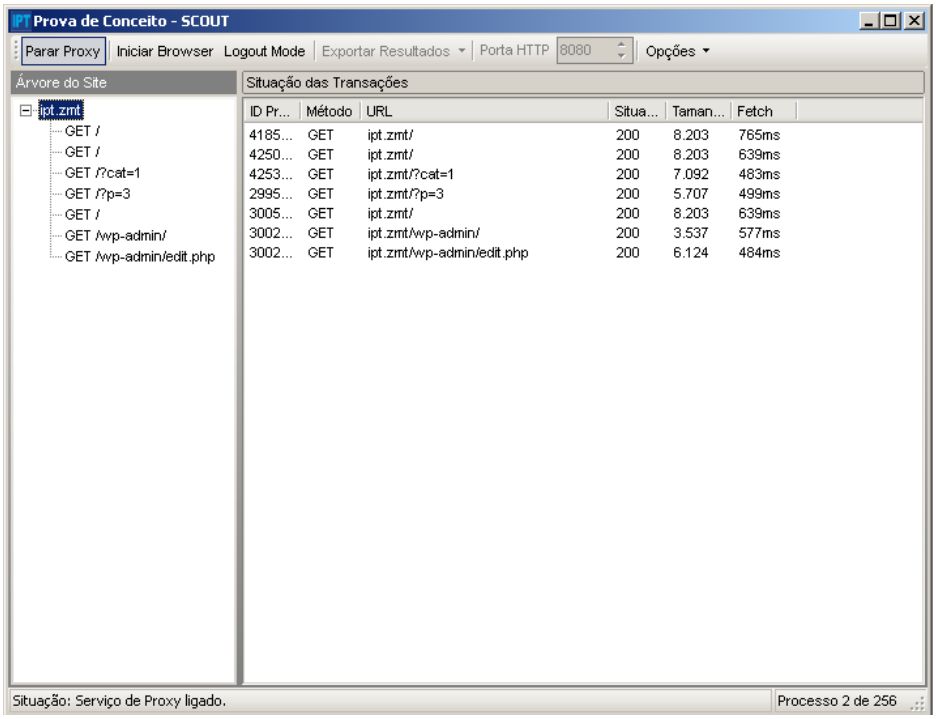

Figura 5 – Ilustração da ferramenta de implementação do SCOUT

Fonte: Elaborado pelo autor.

De acordo com as atividades descritas pelo SCOUT no capítulo 4, os seguintes requisitos funcionais foram utilizados para implantação da ferramenta:

a) Uma interface para captura dos requisitos a ser utilizada nos testes, representada pela tabela 1 (Atividade 1);

b) Procedimento de rastreamento e validação automática dos requisitos (Atividade 1);

c) Uma interface para configuração dos casos de uso primários e secundários (Atividade 1);

d) Uma interface para a construção e armazenamento dos padrões e regras de transformação para os casos de teste por meio de *scripts* (Atividade 2);

e) Procedimento para gestão dos *tokens* de controle de sessão de usuário (Atividade 3);

f) Procedimento para coleta de URL por meio da execução dos casos de uso primários e secundários, representada pela tabela 2 (Atividade 4);

g) Procedimento para interpretação dos *scripts* e execução dos casos de testes (Atividade 5 e Atividade 6);

h) Procedimento para geração de relatórios (Atividade 7).

Requisitos	Valor
R1	http://teste.ipt.br, http://loginteste.ipt.br
R2	**Autenticação de Usuário**: Entrar no formulário de login; utilizar usuário teste e senha teste; escolher a opção de teste. **Consulta de notas**: Escolher a opção consulta de notas; entrar com a matrícula 999999-1; clicar em extrato anual.
R3	JSESSIONID, SESSIONID
R4	1) Autenticação de Usuário; 2) Consulta de Notas.
R5	Cross-site scripting (A2)

Tabela 1 – Exemplo da Planilha de requisitos para o teste de segurança

Método	URL	POST
GET	http://teste.ipt.br/index.php	N/A
POST	http://teste.ipt.br/login.php	user=teste&senha=teste
GET	http://teste.ipt.br/menu.php?o=1	N/A
GET	http://teste.ipt.br/extrato.php	N/A
POST	http://teste.ipt.br/extrato.php	matricula=9999-1&tipo=anual

Tabela 2 – Exemplo de planilha descrevendo URL capturadas

5.5.2 Método dos testes de caixa-preta

Para a execução dos testes de segurança de caixa-preta (BAU et al., 2010) foi utilizada a ferramenta de código aberto *Web Application Attack and Audit Framework* (w3af), disponível gratuitamente e amplamente utilizada para testes de segurança em aplicativos web. A plataforma de instalação foi o ambiente operacional Windows 32-bits.

Outras ferramentas também foram analisadas e consideradas para a implantação dos testes de caixa preta, entretanto, por se tratar de aplicativos comerciais, não estavam integralmente disponíveis para realização dos testes em ambiente privativo e controlado. São elas:

- Cenzic;
- N-Stalker;
- HP WebInspect.

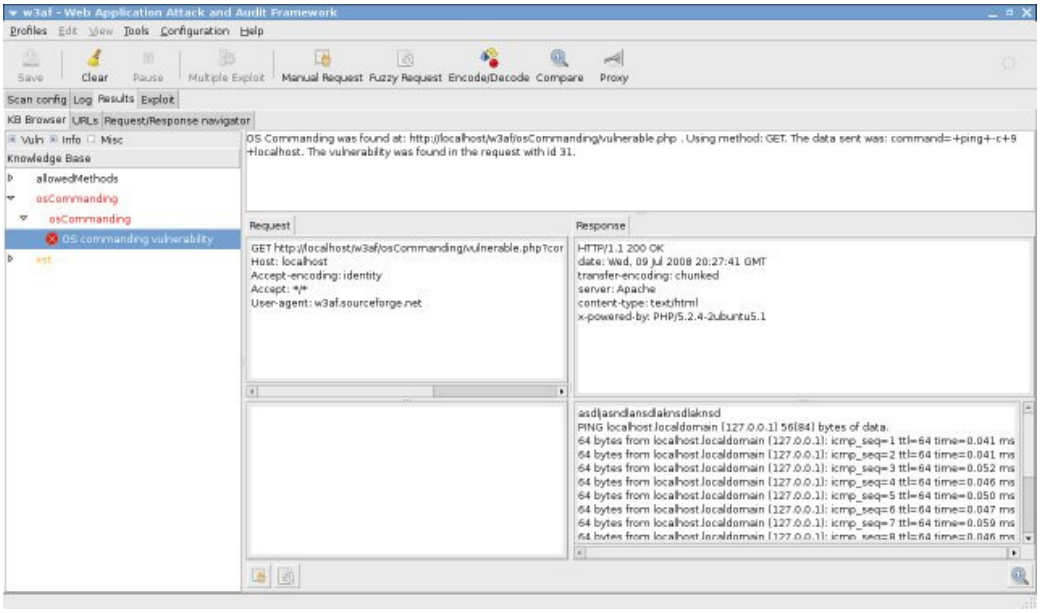

Figura 6 – Ilustração da ferramenta para testes de caixa-preta *w3af*.

Fonte: Elaborado pelo autor.

5.5.3 Método SecuBat

Para a execução do SecuBat (KALS et al., 2006) foi utilizada a ferramenta de prova de conceito desenvolvida pelos próprios autores, construída sob a tecnologia Microsoft.NET. A plataforma de instalação foi o ambiente operacional Windows 32-bits com suporte para uma instância de banco de dados hospedada em tecnologia Microsoft SQL.

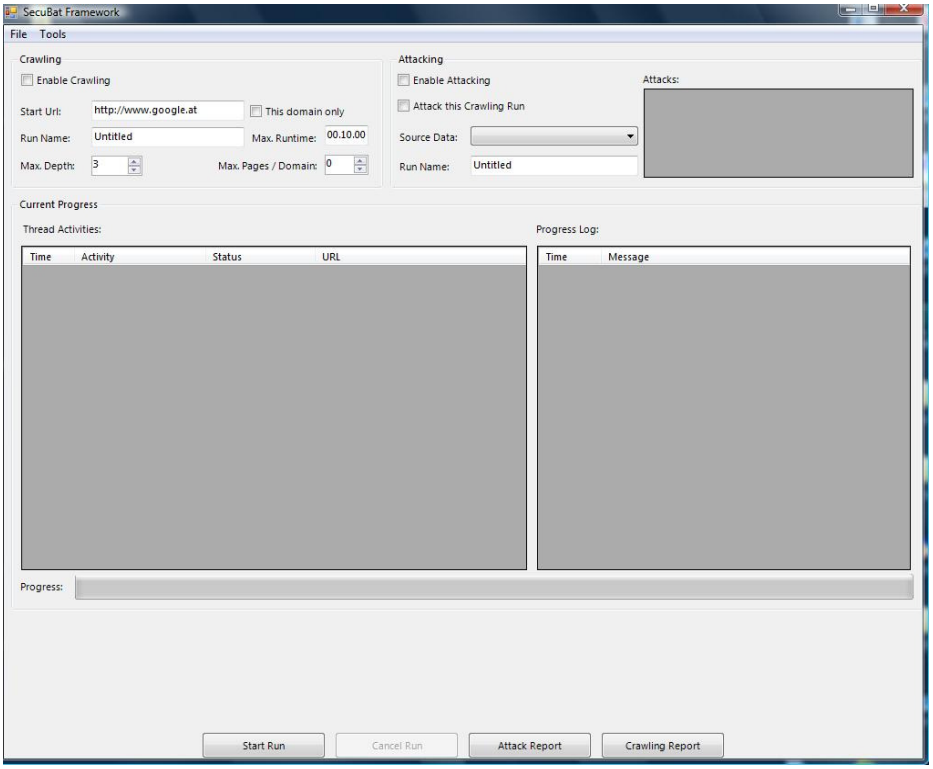

Figura 7 – Ilustração da ferramenta para testes do SecuBat.

Fonte: Elaborado pelo autor.

5.6 Testes a serem executados

Para a execução dos testes, foram escolhidas classes de vulnerabilidade que apresentem grande incidência, nível simplificado de exploração e detecção; e, principalmente, que sejam também utilizados pelos outros métodos descritos no estado da arte.

Para se identificar os candidatos, foi utilizada a pesquisa OWASP Top 10 (2010), descrita na seção 2.3.1 e referenciada de maneira sumarizada na tabela 3, que comprove informações relativas à severidade e nível de complexidade para exploração. Para se encontrar o nível de incidência destas falhas, utilizou-se o guia *Common Weakness Enumeration* (MITRE, 2011), que fornece uma lista das vulnerabilidades mais comuns e frequentes dos aplicativos em geral. As classes de vulnerabilidades escolhidas são:

a) *Cross-site scripting* (A2);

b) Injeção de SQL (A1);

c) Injeção de comandos arbitrários no sistema operacional (A1);

Classe	Incidência	Severidade	Exploração	Detecção
A1	**Média**	Alta	**Média**	Complexa
A2	**Alta**	Média	**Simples**	**Simples**
A3	Média	Alta	Média	Média
A4	Média	Média	Simples	Simples
A5	Alta	Média	Média	Simples
A6	Média	Média	Simples	Simples
A7	Média	Alta	Simples	Complexa
A8	Baixa	Média	Simples	Média
A9	Média	Média	Complexa	Simples
A10	Baixa	Média	Simples	Simples

Tabela 3 – Análise das classes de vulnerabilidade OWASP Top 10

De acordo com a pesquisa realizada pela Cenzic (2010), mais da metade dos problemas críticos encontrados em aplicativos web estavam ligados a uma destas classes de vulnerabilidades. As principais consequências destas vulnerabilidades são o roubo de identidade, por meio da aquisição dos *tokens* de controle de sessão, roubo de informação dos bancos de dados e, em alguns casos, a execução arbitrária de comandos no sistema operacional do servidor ou no navegador web do cliente legítimo do aplicativo.

O *cross-site scripting* e a injeção de SQL também são testados nos métodos WAVES (HUANG et al., 2004), SecuBat (KALS et al., 2006), ARDILLA (KIEZUN et al., 2006) e D-WAV (ZHANG et al., 2010), contudo, o SCOUT se propõe a estender os testes e permitir que novas classes de vulnerabilidade sejam testadas em implementações futuras.

5.7 Resultado dos testes

Nesta seção se apresenta o resultado dos testes executados por cada um dos métodos. Para melhor demonstrar o caminho percorrido no aplicativo web, serão descritas as URL utilizadas para a execução de cada um dos casos de uso.

Para fins de melhor comparação do grau de assertividade de cada um dos métodos, foram descritas também as vulnerabilidades relativas aos casos de uso escolhidos para os testes, de acordo com os dados do NVD.

5.7.1 Caso de uso primário 1: Autenticação do usuário

As transações que compreendem este caso de uso são:

a) http://blog:80/ (Método Get)

b) http://blog:80/wp-login.php (Método Get)

c) http://blog:80/wp-login.php (Método Post). Dados enviados: log=admin&pwd=ipt&submit=Login+%C2%BB&redirect_to=wp-admin%2F

d) http://blog:80/wp-admin/ (Método Get)

A vulnerabilidade existente para esse caso de uso:

- **CVE-2004-1559**: Múltiplas vulnerabilidades de *cross-site scripting*;

5.7.1.1 Resultado dos testes de caixa-preta

As URL percorridas pelo método foram “a”, “b” e “c”. Como o método não permite a configuração de uma credencial válida para o acesso à área administrativa, não foi possível encontrar e percorrer a URL “d”.

Como a única vulnerabilidade existente estava ligada a uma URL que não necessitava de autenticação (CVE-2004-1559), o método encontrou corretamente o problema deste caso de uso por meio da seguinte requisição:

- **Requisição “c” via método Post:** http://blog:80/wp-login.php
  - **Dados enviados:**
    log=admin&pwd=ipt&submit=Login+%C2%BB&redirect_to=
    <script>alert(‘XSS’)</script>

- **Resultado:** Foi apresentada uma janela com o conteúdo "XSS" após a execução desta transação, conforme vulnerabilidade especificada em CVE-2004-1559.

5.7.1.2 Resultado do SecuBat

Assim como Bau et al., as URLs percorridas pelo método foram "a", "b" e "c". Como o método não permite a configuração de uma credencial válida para o acesso à área administrativa, não foi possível encontrar e percorrer a URL "d".

Por se tratar de uma vulnerabilidade existente em uma variável de formulário web no arquivo wp-login.html (CVE-2004-1559), o método detectou corretamente a única vulnerabilidade existente por meio da seguinte requisição:

- **Requisição "c" via método Post:** http://blog:80/wp-login.php
  - **Dados enviados:** log=admin&pwd=ipt&submit=Login+%C2%BB&redirect_to=<script>alert('XSS')</script>
  - **Resultado:** Foi apresentada uma janela com o conteúdo "XSS" após a execução desta transação, conforme vulnerabilidade CVE-2004-1559.

5.7.1.3 Resultado do SCOUT

Diferentemente dos outros métodos, o SCOUT fornece credenciais de acesso previamente gravadas por meio da execução dos casos de uso e mantém a persistência da sessão. Desta forma, todas as URL foram percorridas com sucesso.

O método também encontrou corretamente a única vulnerabilidade existente neste caso de uso (CVE-2004-1559) por meio da seguinte requisição:

- **Requisição "c" via método Post:** http://blog:80/wp-login.php

- **Dados enviados:**
  log=admin&pwd=ipt&submit=Login+%C2%BB&redirect_to=<**IPT**><h1>TesteXSS
- **Resultado:** Foi localizada uma entidade HTML de nome "IPT" após a execução desta transação, confirmando a vulnerabilidade de injeção HTML especificada no CVE-2004-1559.

5.7.1.4 Possíveis resultados dos outros métodos

Tanto o método WAVES quanto o D-WAV utilizam o mesmo mecanismo do SecuBat para detectar pontos de entradas de dados nas transações HTTP, portanto, a vulnerabilidade especificada em CVE-2004-1559 também poderia ter sido facilmente detectada.

5.7.1.5 Conclusão dos resultados para o caso de uso 1

De acordo com os resultados para o caso de uso 1 (quadro 2), infere-se que todos os métodos testados apresentam alto nível de detecção de vulnerabilidades quando as interações não exigem autenticação e persistência de sessão do usuário.

Observou-se, porém, que o SCOUT foi o único método que efetivamente realizou a autenticação do usuário de acordo com o caso de uso apresentado, obtendo um *token* de sessão válido para o sistema.

Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2004-1559 (sem sessão)	**Sim**	**Sim**	**Sim**

Quadro 2 – Sumário dos resultados para o caso de uso 1
Fonte: Elaborado pelo autor.

5.7.2 Caso de uso secundário 2: Listagem de artigos

As principais transações que compreendem este caso de uso são:

a) http://blog:80/ (Método Get)

b) http://blog:80/?cat=1 (Método Get)

c) http://blog:80/?p=2 (Método Get)

d) http://blog:80/?p=1 (Método Get)

A vulnerabilidade existente neste caso de uso:

- **CVE-2005-1810**: Injeção SQL na área de categorias;

5.7.2.1 Resultado dos testes de caixa-preta

Por não exigir autenticação ou quaisquer outros tipos de interações específicas, o teste de caixa-preta percorreu com sucesso todas as URLs que compõe este caso de uso. O método também encontrou corretamente a vulnerabilidade CVE-2005-1810 deste caso de uso por meio da seguinte requisição:

- **Requisição "b" via método Get:** http://blog:80/?cat=1
  - **Dados enviados:** http://blog:80/?cat=1' AND '1'='1
  - **Resultado:** A ferramenta detectou corretamente a existência de uma injeção de SQL na variável "cat" conforme descrito no CVE-2005-1810.

5.7.2.2 Resultado do SecuBat

Da mesma forma que os testes de caixa-preta, o SecuBat navegou por todas as URLs sem dificuldade, contudo, por não existir nenhuma interação por meio de formulário web, não foram encontrados pontos de entrada de dados para a execução de ataques.

Desta forma, o método SecuBat não detectou corretamente a vulnerabilidade existente neste caso de uso e especificada no CVE-2005-1810.

5.7.2.3 Resultado do SCOUT

O método SCOUT navegou corretamente por todas as URLs que compõem este caso de uso. Durante toda execução deste fluxo, o estado de sessão do usuário autenticado foi mantido. O SCOUT também encontrou corretamente a vulnerabilidade CVE-2005-1810 deste caso de uso por meio da seguinte requisição:

- **Requisição "b" via método Get:** http://blog:80/?cat=1
  - **Dados enviados:** http://blog:80/?cat='%20TesteIPT
  - **Resultado:** A ferramenta detectou corretamente a existência de uma injeção de SQL na variável "cat" conforme descrito no CVE-2005-1810.

5.7.2.4 Possíveis resultados dos outros métodos

Por utilizarem o mesmo processo de captura dos pontos de entrada de dados via formulário web do SecuBat, os métodos WAVES e D-WAV também teriam dificuldade em detectar a existência da vulnerabilidade especificada no CVE-2005-1810.

5.7.2.5 Conclusão dos resultados para o caso de uso 2

Os resultados obtidos a partir dos testes para o caso de uso 2 (quadro 3) permitem inferir que o método empregado pelo SecuBat, a captura de pontos de entrada do aplicativo web por meio de formulários web, apresenta baixa assertividade nas transações que não exigem interações diretas com o usuário. Outros métodos como WAVES e D-WAV também empregam as mesmas técnicas, portanto, são suscetíveis ao mesmo problema.

Com relação aos métodos de teste de caixa-preta e SCOUT, o grau de assertividade foi satisfatório. As transações do caso de uso 2 não exigiam credenciais válidas e uso de sessão válida no sistema, contudo, o método SCOUT permaneceu com a persistência da sessão obtida a partir do caso de uso 1.

Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2005-1810 (sem sessão)	**Sim**	*Não*	**Sim**

Quadro 3 – Sumário dos resultados para o caso de uso 2
Fonte: Elaborado pelo autor.

5.7.3 Caso de uso secundário 3: Publicação de artigo

As principais transações que compreendem este caso de uso são:

a) http://blog:80/ (Método Get)

b) http://blog:80/wp-admin/ (Método Get)

c) http://blog:80/wp-admin/edit.php (Método Get)

d) http://blog:80/wp-admin/post.php (Método Get)

e) http://blog:80/wp-admin/post.php (Método Post). Dados Enviados: user_ID=1&action=post&post_title=Testando+Post+Web&post_category%5B%5D=1&content=Mas%2C+%C3%A0+primeira+vista%2C+qui%C3%A7%C3%A1+pare%C3%A7a+que+a+consolida%C3%A7%C3%A3o+das+estruturas+psicol%C3%B3gicas+nos+obriga+%C3%A0+an%C3%A1lise+das+condi%C3%A7%C3%B5es+epistemol%C3%B3gicas+e+cognitivas+exigidas.+Por+outro+lado%2C+a+complexidade+dos+estudos+efetuados+%C3%A9+uma+das+consequ%C3%AAncias+dos+relacionamentos+verticais+entre+as+hierarquias+conceituais.+&post_pingback=1&trackback_url=&publish=Publish&referredby=http%253A%252F%252Fblog%252Fwp-admin%252Fedit.php

f) http://blog:80/wp-admin/post.php?posted=true (Método Get)

g) http://blog:80/ (Método Get)

A vulnerabilidade existente para esse caso de uso:

- **CVE-2005-1102**: *Cross-site scripting* na seção de adição de post;

5.7.3.1 Resultado dos testes de caixa-preta

A execução deste caso de uso exige necessariamente a apresentação de credenciais válidas no painel de administração do *wordpress*, realizadas por meio do caso de uso 1, assim como a persistência de sessão do usuário autenticado. Nestas condições, o método de testes de caixa-preta realizou a navegação tão somente pelas URLs “a” e “b”, não sendo possível detectar a vulnerabilidade especificada em CVE-2005-1102, localizada na URL ‘e’.

5.7.3.2 Resultado do SecuBat

Assim como os testes de caixa-preta, o SecuBat não apresenta credenciais válidas e não possui a capacidade de manter a persistência do estado de sessão de um usuário válido, portanto, apenas as URLs “a” e “b” foram acessadas. A vulnerabilidade especificada no CVE-2005-1102 igualmente não foi detectada.

5.7.3.3 Resultado do SCOUT

O SCOUT possui a capacidade de fornecer as variáveis necessárias para garantir que a navegação ocorra em todas as URLs do caso de uso, incluindo a persistência do estado de sessão de um usuário autenticado. Desta forma, o método executou todas as URL descritas neste caso de uso.

O SCOUT também detectou corretamente a vulnerabilidade existente neste caso de uso (CVE-2005-1102), acessível somente aos usuários autenticados no painel de administração:

- **Requisição “e” via método Post:** http://blog:80/wp-admin/post.php
    - **Dados enviados:**
      user_ID=1&action=post&post_title=**<IPT>**<h1>Teste XSS&post_category%5B%5D=1&content=Mas%2C+%C3%A0+primeira+vista%2C+qui%C3%A7%C3%A1+pare%C3%A7a+que+a+consolida%C3%A7%C3%A3o+das+estruturas+psico-l%C3%B3gicas+nos+obriga+%C3%A0+an%C3%A1lise+das+condi%C3%A7%C3%B5es+epistemol%C3%B3gicas+e+cognitivas+exigidas.+Por+outro+lado%2C+a+complexidade+dos+estudos+efetuados+%C3%A9+uma+das+consequ%C3%AAncias+dos+relacionamentos+verticais+entre+as+hierarquias+conceituais.+&post_pingback=1&trackback_url=&publish=Publish&referredby=http%253A%252F%252Fblog%252Fwp-admin%252Fedit.php
    - **Resultado:** A ferramenta detectou a existência da entidade HTML “IPT” no artigo recém-publicado, conforme especificado na vulnerabilidade CVE-2005-1102.

5.7.3.4 Possíveis resultados dos outros métodos

Da mesma forma que SecuBat, os métodos WAVES e D-WAV também não realizam o processo de autenticação e não mantém a persistência do estado de sessão de um usuário, portanto, não conseguiriam detectar a existência da vulnerabilidade especificada no CVE-2005-1102.

5.7.3.5 Conclusão dos resultados para o caso de uso 3

Este caso de uso provoca uma comparação interessante entre os diferentes métodos, uma vez que exige credenciais válidas no sistema e a persistência da sessão obtida durante o processo de autenticação (*token*).

Diferentemente do SCOUT, todos os métodos utilizados para comparação não especificam e utilizam o contexto dos casos de uso, incluindo autenticação de usuários, para garantir que transações como a do caso de uso 4 sejam executadas de maneira eficaz. O resultado, como indica o quadro 4, é um baixo grau de assertividade.

Em relação ao SCOUT, a utilização do contexto dos casos de uso e a persistência de sessão em cada um dos seus testes é uma vantagem técnica que aumenta de maneira considerável a assertividade do nível de detecção de vulnerabilidades.

Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2005-1102 (**com** sessão)	*Não*	*Não*	**Sim**

Quadro 4 – Sumário dos resultados para o caso de uso 3
Fonte: Elaborado pelo autor.

5.7.4 Caso de uso secundário 4: Publicação de comentário

As principais transações que compreendem este caso de uso são:

a) http://blog:80/?p=3 (Método Get)

b) http://blog:80/wp-comments-post.php (Método Post). Dados enviados:
author=teste&email=teste@teste.ipt.br&url=http%3A%2F%2Fteste&comment=teste+do+coment%C3%A1rio+IPT&submit=Submit+Comment&comment_post_ID=3

c) http://blog:80/?p=3 (Método Get)

As vulnerabilidades existentes para este caso de uso:

- **CVE-2006-0985**: *Cross-site scripting* na inserção de comentários;
- **CVE-2006-1012**: Injeção de SQL por meio de variáveis do cabeçalho HTTP.

5.7.4.1 Resultado dos testes de caixa-preta

Por não exigir nenhum tipo de autenticação ou persistência de estado de sessão, os testes de caixa-preta navegaram por todas as URL deste caso de uso. Da mesma forma, as duas vulnerabilidades existentes nestas transações (CVE-2006-0985 e CVE-2006-1012) foram corretamente detectadas:

- **Requisição "b" via método Post:** http://blog:80/wp-comments-post.php
  - **Dados enviados:** author=<script>alert('XSS')</script>&email=<script>alert('XSS')</script>&url=<script>alert('XSS')</script>&comment=<script>alert('XSS')</script>&submit=Submit+Comment&comment_post_ID=3
  - **Resultado:** Foi detectada a presença de um *script* arbitrário que abriu uma janela contendo a palavra "XSS", configurando a vulnerabilidade de *cross-site scripting* conforme descrito no CVE-2006-0985.
- **Requisição "b" via método Post:** http://blog:80/wp-comments-post.php
  - **Dados enviados:** author=1' AND '1'='1&email=1' AND '1'='1&url=1' AND '1'='1&comment=1' AND '1'='1&submit=Submit+Comment&comment_post_ID=3
  - **Cabeçalho HTTP:** User-Agent: 1' AND '1'='1
  - **Resultado:** A ferramenta detectou corretamente a existência de uma injeção SQL na variável User-Agent do cabeçalho HTTP, conforme descrito no CVE-2006-1012.

5.7.4.2 Resultado do SecuBat

Assim como os testes de caixa-preta, o SecuBat não teve dificuldade de navegar por todas as URLs deste caso de uso. Contudo, por estar restrito à

detecção de pontos de entrada de dados em formulários web, o método identificou apenas a vulnerabilidade especificada em CVE-2006-0985, falhando ao identificar a CVE-2006-1012 que envolve a modificação de variáveis do cabeçalho HTTP:

- **Requisição "b" via método Post:** http://blog:80/wp-comments-post.php
  - **Dados enviados:** author=<script>alert('XSS')</script>&email=<script>alert('XSS')</script>&url=<script>alert('XSS')</script>&comment=<script>alert('XSS')</script>&submit=Submit+Comment&comment_post_ID=3
  - **Resultado:** Foi detectada a presença de um *script* arbitrário que abriu uma janela contendo a palavra "XSS", configurando a vulnerabilidade de *cross-site scripting* conforme descrito no CVE-2006-0985.

5.7.4.3 Resultado do SCOUT

O SCOUT navegou por todas as URL da transação e manteve, durante toda a interação, o estado de sessão do usuário autenticado. Em razão das características específicas da vulnerabilidade detalhada em CVE-2006-0985, que exige que o usuário não esteja autenticado no sistema, o SCOUT só detectou corretamente a vulnerabilidade nas variáveis do cabeçalho HTTP (CVE-2006-1012):

- **Requisição "b" via método Post:** http://blog:80/wp-comments-post.php
  - **Dados enviados:** author='%20TesteIPT&email='%20TesteIPT&url='%20TesteIPT&comment='%20TesteIPT&submit=Submit+Comment&comment_post_ID=3
  - **Cabeçalho HTTP:** User-Agent: '%20TesteIPT

- **Resultado:** A ferramenta detectou corretamente a existência de uma injeção SQL na variável User-Agent do cabeçalho HTTP, conforme descrito no CVE-2006-1012.

5.7.4.4 Possíveis resultados dos outros métodos

Da mesma forma que SecuBat, os métodos WAVES e D-WAV também estão restritos aos pontos de entrada de formulários web, portanto, teriam provável dificuldade em detectar a vulnerabilidade que envolve a modificação de variáveis do cabeçalho HTTP (CVE-2006-1012). Com relação à vulnerabilidade especificada em CVE-2006-0985, infere-se que os métodos não teriam dificuldade em detectá-la, uma vez que também não utilizam nenhum método de autenticação e persistência de sessão.

5.7.4.5 Conclusão dos resultados para o caso de uso 4

Este caso de uso compreende transações com duas vulnerabilidades completamente distintas e com o objetivo de testar exclusivamente pontos de entrada de dados pelo cabeçalho HTTP e a gerência do estado de sessão de maneira apropriado.

Com relação à vulnerabilidade especificada em CVE-2006-0985, por se tratar de uma falha que só seria passível de exploração nos casos que o usuário não estivesse autenticado e, portanto, sem apresentar um *token* válido de sessão, o SCOUT apresentou ineficácia de detecção (quadro 5). Já em relação aos outros métodos, sem quaisquer técnicas para persistência de sessão, o problema foi detectado com grande assertividade.

Para a vulnerabilidade que pressupõe a obtenção dos pontos de entrada por meio do cabeçalho HTTP (CVE-2006-1012), percebe-se novamente que as técnicas utilizadas pelos métodos SecuBat, WAVES e D-WAV apresentam baixo nível de detecção para estes cenários (quadro 5), uma vez que utilizam como referência apenas os pontos expostos por meio de formulários web.

Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2006-0985 (sem sessão)	**Sim**	**Sim**	*Não*
CVE-2006-1012 (sem sessão)	**Sim**	*Não*	**Sim**

Quadro 5 – Sumário dos resultados para o caso de uso 4
Fonte: Elaborado pelo autor.

5.7.5 Caso de uso secundário 5: Remoção de comentário

As principais transações que compreendem este caso de uso são:

a) http://blog:80/wp-admin/ (Método Get)

b) http://blog:80/wp-admin/edit.php (Método Get)

c) http://blog:80/wp-admin/edit-comments.php (Método Get)

d) http://blog:80/wp-admin/post.php?action=deletecomment&p=3&comment=5 (Método Get)

e) http://blog:80/wp-admin/edit-comments.php (Método Get)

A vulnerabilidade existente para esse caso de uso:

- **CVE-2005-2107**: *Cross-site scripting* na remoção de comentários;

5.7.5.1 Resultado dos testes de caixa-preta

Este caso de uso envolve acesso ao painel administrativo do *wordpress* e exige a persistência do estado de sessão de um usuário autenticado. Desta forma, os testes de caixa-preta não conseguiram executar as URL e detectar a vulnerabilidade especificada em CVE-2005-2107.

5.7.5.2 Resultado do SecuBat

Da mesma forma que os testes de caixa-preta, o SecuBat não consegue persistir o estado de sessão de um usuário autenticado e, portanto, não executou as URL do caso de uso, não detectando a vulnerabilidade especificada em CVE-2005-2107.

5.7.5.3 Resultado do SCOUT

O SCOUT navegou por todas as URLs da transação com o estado de sessão do usuário autenticado. Durante a execução do caso de uso, foi detectada a vulnerabilidade CVE-2005-2107 existente no processo de remoção de comentários:

- **Requisição "d" via método Get:** http://blog:80/wp-admin/post.php?action=deletecomment&p=3&comment=5
  - **Dados enviados:** http://blog:80/wp-admin/post.php?action=deletecomment&p=3&comment=%22%3E%3CIPT%3E%3Ch1%3ETESTEXSS ou http://blog:80/wp-admin/post.php?action=deletecomment&p=%22%3E%3CIPT%3E%3Ch1%3ETESTEXSS&comment=2
  - **Resultado:** A ferramenta detectou corretamente a existência de uma entidade HTML "IPT" na resposta, configurando um *cross-site scripting* conforme descrito no CVE-2005-2107.

5.7.5.4 Possíveis resultados dos outros métodos

Da mesma forma que SecuBat, os métodos WAVES e D-WAV também não possuem capacidade de persistir o estado de sessão de um usuário autenticado, desta forma, não conseguem acessar os pontos de entrada de dados do formulário web para construção dos casos de teste e, por conseguinte, detectar a vulnerabilidade descrita em CVE-2005-2107.

5.7.5.5 Conclusão dos resultados para o caso de uso 5

Os resultados do caso de uso 5 (quadro 6) consolidam a inferência que, quando as transações testadas no aplicativo web pressupõem o uso de autenticação e persistência de sessão válida, todos os métodos, com exceção do SCOUT, possuem baixa assertividade no nível de detecção de vulnerabilidades.

As técnicas empregadas pelo SCOUT, incluindo o uso contextual dos casos de uso em todos os testes e a persistência da sessão válida do usuário (*token)*, permitem que transações mais específicas e inacessíveis aos outros métodos sejam testadas de maneira eficaz.

Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2005-2107 (**com** sessão)	*Não*	*Não*	**Sim**

Quadro 6 – Sumário dos resultados para o caso de uso 5
Fonte: Elaborado pelo autor.

5.8 Conclusão do capítulo 5

A validação do SCOUT foi realizada por meio de um estudo de prova de conceito em ambiente experimental de testes com outros métodos apresentados neste trabalho. O aplicativo web utilizado para os testes de segurança foi o *wordpress* (versão 1.5), um sistema de *blogs* popular, gratuito e com um histórico conhecido e comprovado de vulnerabilidades.

Os métodos utilizados para comparação com o SCOUT foram os testes de segurança de caixa-preta (BAU et al., 2010), SecuBat (KALS et al., 2006), WAVES (HUANG et al., 2004) e D-WAV (ZHANG et al., 2010). As ferramentas para os métodos de testes de caixa-preta e SecuBat foram implantadas em sistema operacional independente e com variáveis similares para garantia da independência dos resultados. Os métodos WAVES e D-WAV, por não dispor de ferramentas abertas para testes, foram executados de maneira teórica, entretanto, preservando as premissas estabelecidas pelos autores.

Para determinar a eficácia de cada um dos métodos e traçar um quadro comum de comparação, foi utilizado o critério de assertividade na detecção de vulnerabilidades existentes, de acordo com o banco de dados do NVD (NIST). Os seguintes casos de uso mais comuns, de acordo com o manual de utilização do aplicativo, foram utilizados como referência:

a) Caso de uso 1: Autenticação do Usuário;

b) Caso de uso 2: Listagem de artigos;

c) Caso de uso 3: Publicação de artigo;

d) Caso de uso 4: Publicação de comentário;

e) Caso de uso 5: Remoção de comentário.

Para eleger as classes de vulnerabilidades testadas, utilizou-se como referência o guia *Common Weakness Enumeration* (MITRE, 2011) e a pesquisa da Cenzic (2010), que esclarecem que as seguintes vulnerabilidades são as mais frequentes nos aplicativos web:

a) *Cross-site scripting* (A2);

b) Injeção de SQL (A1);

c) Injeção de comandos arbitrários no sistema operacional (A1).

A partir da implantação do ambiente experimental, os testes foram conduzidos de acordo com as especificações de cada um dos métodos testados, utilizando-se como referência as transações dos casos de uso especificados, lista de vulnerabilidades do NVD e o nível de detecção apresentado.

O resultado final, ilustrado no quadro 7, demonstra que o SCOUT obteve 83% de assertividade no nível de detecção de vulnerabilidades, seguido de 66% para os testes de caixa-preta e 33% para o SecuBat.

Resumo dos Resultados da Validação da Proposta			
Caso de Uso	BAU et al.	SecuBat, WAVES e D-WAV	**SCOUT**
Caso de Uso 1 (autenticação)	1/1	1/1	1/1
Caso de Uso 2 (sem autenticação)	1/1	0/1	1/1
Caso de Uso 3 (autenticação)	0/1	0/1	1/1
Caso de Uso 4 (sem autenticação)	2/2	1/2	1/2
Caso de Uso 5 (autenticação)	0/1	0/1	1/1
Assertividade	**66%**	**33%**	**83%**

Quadro 7 – Resumo de Resultados da Validação da Proposta
Fonte: Elaborado pelo autor.

Os testes demonstraram que, de acordo com o quadro 8, o SCOUT apresenta maior assertividade nas transações dos casos de uso que tenham as seguintes características:

a) Requerer apresentação de credenciais válidas para autenticação (CVE-2006-1102 e CVE-2006-2107);

b) Requerer persistência do estado de sessão válido do usuário ao longo dos testes (CVE-2006-1102 e CVE-2006-2107).

Estas características não são exploradas de maneira eficiente dentro dos outros métodos comparados, uma vez que não pressupõem o requerimento da utilização do contexto dos casos de uso em todos os testes, como a autenticação do usuário e, igualmente, não persistem à sessão válida ao longo dos testes.

Resumo dos Resultados por Vulnerabilidade Detectada			
Vulnerabilidade	BAU et al.	SecuBat	**SCOUT**
CVE-2004-1559 (sem sessão)	**Sim**	**Sim**	**Sim**
CVE-2005-1810 (sem sessão)	**Sim**	*Não*	**Sim**
CVE-2005-1102 (com sessão)	*Não*	*Não*	**Sim**
CVE-2006-0985 (sem sessão)	**Sim**	**Sim**	*Não*
CVE-2006-1012 (sem sessão)	**Sim**	*Não*	**Sim**
CVE-2005-2107 (com sessão)	*Não*	*Não*	**Sim**

Quadro 8 – Resumo dos Resultados por Vulnerabilidade Detectada
Fonte: Elaborado pelo autor.

Por outro lado, estas características também podem dificultar a detecção de situações bastante especiais, como o caso da vulnerabilidade especificada em CVE-2006-0985, que tem por requerimento o fato de não estar autenticado no sistema. Com exceção do SCOUT, todos os métodos tiveram sucesso na detecção.

Os testes de caixa-preta (BAU et al., 2010) demonstraram um grau satisfatório de 66% de assertividade (quadro 8) na detecção de vulnerabilidades, desde que as transações não exijam conhecimento prévio do responsável pelos testes de segurança. O método falha em situações em que o SCOUT teve sucesso, como no acesso ao painel administrativo do caso de uso 5, por não tratar da utilização de casos e persistência da sessão do usuário ao longo de todos os testes.

De todos os métodos testados, o SecuBat foi o que apresentou o resultado mais baixo, com um nível de 33% de assertividade (quadro 8). O método teve dificuldade de detecção de vulnerabilidades em razão das seguintes características:

a) Método utiliza como referência de pontos de entrada do aplicativo web apenas os formulários (CVE-2005-1810 e CVE-2006-1012);

b) Não emprega o uso de contexto e persistência de sessão ao longo de todos os testes (CVE-2005-1102 e CVE-2005-2107).

Apesar de não terem sido testados com o emprego de ferramenta, os métodos WAVES e D-WAV utilizam o mesmo referencial técnico para obtenção dos pontos de entrada e persistência de sessão, portanto, também seriam ineficazes por igual.

Conclui-se, portanto, por meio dos resultados obtidos, que não existem métodos que apresentem eficácia absoluta na detecção de vulnerabilidades. Por meio das premissas propostas pelo método SCOUT, observa-se que a utilização do conhecimento prévio dos casos de uso específicos do aplicativo web, incluindo as características exclusivas de interação e persistência do estado de sessão do usuário, pode contribuir de maneira qualitativa para o aumento da assertividade dos testes de segurança nos aplicativos web.

Por final, o estudo permite afirmar que um das melhores maneiras de aumentar consideravelmente a assertividade dos testes de segurança dentro do SDLC é por meio do uso combinado de diferentes métodos, seja para as transações mais simples que não pressupõe autenticação e persistência do estado de sessão ou para aquelas que fazem algum uso destas características.

6 CONCLUSÕES

Práticas para o desenvolvimento de aplicações mais seguras já foram estabelecidas há mais de uma década, contudo, falhas de segurança continuam sendo reportadas freqüentemente em pleno estágio de produção dos aplicativos web. Apesar dos principais modelos de segurança especificarem atividades de testes de segurança para detecção e correção de vulnerabilidades no ciclo de desenvolvimento, a falta de emprego do conhecimento das funcionalidades do aplicativo dificulta a elaboração de requisitos mais eficientes de segurança e, consequentemente, testes mais eficazes para detecção de vulnerabilidades.

Este trabalho avaliou propostas recentes que tratam da questão da inserção deste conhecimento nos testes, contudo, detectou-se que ainda persistem limitações que podem restringir a adoção e eficácia dos testes de segurança no atual contexto dos aplicativos web. Portanto, a motivação para o desenvolvimento de um novo método se sustentou justamente na oportunidade de oferecer solução para algumas das restrições levantadas, incluindo:

a) A oportunidade de desenvolver um método de testes de segurança sem a necessidade da leitura do código-fonte como requisito de entrada;

b) O uso de conhecimentos funcionais do aplicativo para realização de testes, incluindo interações específicas que não dependam da descoberta automática de transações;

c) O enfoque na integração do método ao SDLC, permitindo a repetição e comparação dos resultados dos testes de maneira sistemática ao longo do ciclo de vida do aplicativo;

d) O emprego da premissa de extensibilidade para permitir a detecção de um número maior de classes de vulnerabilidades.

Desta forma, esse trabalho teve por objetivo a proposição de um método nomeado SCOUT para execução de testes de segurança por meio de casos de uso, permitindo a sua aplicação no contexto de quaisquer aplicativos web.

Para a construção do eixo teórico deste trabalho, se fez necessário estudar importantes conceitos-chave como a definição de falhas de software e suas implicações para a introdução de vulnerabilidades nos sistemas. Em seguida, por meio de padrões existentes, avaliou-se quais atividades seriam necessárias para a execução de testes de segurança sob a perspectiva dos aplicativos web. Igualmente, também foram estudados os mecanismos de comunicação e interação com estes aplicativos, estabelecendo a relação dos testes com o protocolo HTTP, os modelos de uso da persistência da sessão do usuário e os pontos de troca de entrada de dados entre a aplicação e seus utilizadores. Por final, foram levantadas as principais vulnerabilidades mais comuns encontradas nos aplicativos da atualidade e a maneira com que os casos de uso podem ser empregados para testar estas falhas.

Para fins de comparação de resultados, foi realizado um estudo do estado da arte dos testes de segurança com o objetivo de levantar as principais vantagens e desvantagens dos trabalhos recentes. Estudou-se um dos principais métodos empregados na atualidade, o teste de caixa-preta, que, apesar da eficiência para encontrar falhas previamente conhecidas, tem pouca eficácia para as análises que envolvam funcionalidades específicas do aplicativo web. Outros métodos, como teste de caixa-branca e o ARDILLA, tratam das funcionalidades do sistema, contudo, têm a restrição de exigir o acesso e leitura ao código-fonte do aplicativo. Por final, os métodos que apresentavam uma sistemática automática para o descobrimento de funcionalidades do aplicativo, tal como o WAVES, D-WAVE e SecuBat; apresentam restrições quanto ao tipo de interação utilizada para troca de dados e, portanto, sem a possibilidade de empregar casos de usos específicos relevantes como a autenticação de um usuário.

O SCOUT foi proposto como uma série de seis atividades distintas a serem aplicadas sequencialmente, dividindo-se em três grandes fases: Planejamento, que inclui as atividades para levantamento de requisitos e planejamento de execução dos casos de uso; Execução, que compreende as atividades de execução dos casos de usos, preparação dos casos de teste e execução dos testes; e, por final, Resultados, com a atividade de análise e geração de relatórios. A execução do SCOUT pressupõe a entrada de quatro

requisitos básicos, incluindo o endereço do aplicativo web a ser avaliada, descrição dos casos de uso primários e secundários, sequência de execução dos casos de uso e definição das classes de vulnerabilidade para os testes.

A proposta de validação do SCOUT orientou-se na comparação da eficácia dos resultados obtidos em relação aos outros trabalhos estudados no estado da arte, levando em consideração o critério de assertividade, ou seja, nível de detecção de vulnerabilidades. Para proporcionar independência e representatividade, foi escolhido um aplicativo web amplamente utilizado na Internet e com uma base extensa de vulnerabilidades catalogada pelo NIST. Da mesma forma, os testes foram conduzidos em ambiente experimental de prova de conceito sem a interferência de fatores externos.

Para garantir a equivalência na comparação entre os trabalhos, foram escolhidos cinco casos de uso do aplicativo web com o objetivo de que todos os métodos navegassem pelas mesmas transações (URL) correspondentes.

Como resultado da proposta de comparação, observou-se que o SCOUT teve melhor eficácia em termos de assertividade aos outros trabalhos utilizados como referência, apresentando um nível de detecção de vulnerabilidades superior em 20%. Percebeu-se que a principal vantagem do SCOUT em relação aos outros métodos é o uso do conhecimento prévio das funcionalidades do sistema por meio de casos de uso, permitindo a detecção de falhas que residem em áreas do aplicativo que não são normalmente acessadas e avaliadas nos outros testes de segurança. Portanto, a eficácia pode aumentar significativamente de acordo com o tipo de aplicativo e as interações pressupostas por suas transações.

Por outro lado, o SCOUT falhou em detectar vulnerabilidades que pressupõem a ausência de autenticação e uma sessão válida no aplicativo, característica comum dentre os outros métodos executados, e que abre a possibilidade para melhorias futuras em sua implementação.

Conclui-se, portanto, que o trabalho tenha atingido o objetivo de propor um método para execução de testes de segurança em aplicativos web por meio de casos de uso, uma vez que ficou demonstrado o emprego bem sucedido do conhecimento prévio das funcionalidades nos testes. Infere-se, contudo, que

apesar de existirem melhorias significativas auferidas pelo método SCOUT, este não deve substituir o emprego da combinação múltipla das principais vantagens oferecidas por outros métodos disponíveis, garantindo, desta forma, a manutenção da eficiência e, sobretudo, eficácia das atividades de teste de segurança no contexto do SDLC.

As principais contribuições auferidas pelo SCOUT incluem:

- O aumento da eficácia na assertividade da detecção de vulnerabilidades em aplicativos web;
- O emprego de um método com repetição sistemática em diferentes fases do ciclo de desenvolvimento, permitindo a integração e aplicabilidade por todo o SDLC;
- O desenvolvimento de um método genérico que permite a extensibilidade para diferentes tipos de caso de teste, independente das classes de vulnerabilidades a serem avaliadas;
- A oportunidade para aceleração da adoção dos modelos de maturidade de segurança dos aplicativos web por meio do emprego de uma alternativa eficaz para as atividades de testes de segurança.

Alguns fatores de limitação que podem restringir a eficácia deste trabalho e sua possível generalização para outros cenários incluem:

- Dificuldade para especificação dos casos de uso em razão da escassez de documentação nos sistemas. Embora outras fontes não formais como entrevistas com o usuário ou desenvolvedor possam apoiar esta tarefa, a falta de detalhes importantes relativos às funcionalidades do sistema pode vir a comprometer o resultado final;
- A restrição dos testes de validação do trabalho a uma plataforma específica de aplicativos web (*wordpress*), limitando, assim, a extensão dos resultados do SCOUT para um escopo mais genérico.

- O comprometimento da eficácia de detecção de vulnerabilidades em áreas do aplicativo que não exigem a persistência da sessão do usuário. Uma oportunidade para melhoria seria realizar os testes em duas etapas, com e sem contexto de sessão;
- A necessidade de utilização conjunta de outros métodos para garantir a eficácia dos testes de segurança. Por não considerar todas as questões possíveis que envolvam a segurança do aplicativo web, incluindo falhas de infraestrutura e a não utilização de mecanismos de persistência de sessão, o SCOUT deve ser utilizado como complemento a estas atividades.

Os resultados obtidos com este trabalho podem ser complementados extensivamente em propostas futuras, incluindo:

- A proposição de métodos que abordem de maneira prática a questão da automatização dos testes por meio de ferramentas ao longo do SLDC;
- A extensão do método proposto para um contexto mais genérico de aplicativos web, incluindo múltiplas plataformas de desenvolvimento de aplicações;
- A melhoria do método atual para detecção de vulnerabilidades que tenham como particularidade um cenário *ad-hoc*, ou seja, sem a utilização do contexto de sessão válida ou do estado obtido por meio de outros casos de uso;
- Evolução do estudo para medir e melhorar a eficiência dos testes de segurança, levando em consideração fatores de comparação como complexidade e tempo de execução;
- Evolução do estudo da medição da eficácia deste método proposto em conjunto com outros trabalhos existentes, proporcionando elementos comparativos mais precisos para análise custo/benefício e gestão de risco;

- Incorporação de métodos complementares para automatização dos controles defensivos do aplicativo, permitindo a geração automática de filtros de ataque de maneira a garantir sua proteção até que as medidas corretivas sejam implantadas para eliminar as vulnerabilidades encontradas.

REFERÊNCIAS

ARKIN, B. et al. Software Penetration Testing. **IEEE Security and Privacy**, p.84-87, Janeiro-Fevereiro, 2005.

ASSAD, R. et al. **Security Quality Assurance on Web-Based Application through Security Requirements Tests**: Elaboration, Execution and Automation. Fifth International Conference on Software Engineering Advances. Los Alamitos, 2010. p. 272-277.

BAU, J. et al. **State of the Art: Automated Black-Box Web Application Vulnerability Testing**. IEEE Symposium on Security and Privacy, 2010. p.332-345.

BERNERS-LEE, T. et al. **RFC 1738:** Uniform resource locators (URL). Internet Engineering Task Force, 1994. Disponível em http://www.rfc-editor.org/rfc/rfc1738.txt. Acesso em: 24 Abril 2011.

BITTNER, K.; SPENCE, I. **Use Case Modeling**. Addison Wesley Professional, Reading 2002. 34p.

CANFORA, G.; DI PENTA, M. Testing services and service-centric systems: challenges and opportunities. **IT Professional**, vol.8, no.2, Março-Abril 2006. pp.10-17.

CENZIC. **Application Security Trends Report Q3-4 2010**. Disponível em http://www.cenzic.com/resources/reg-not-required/trends/. Acesso em: 24 Abril 2011.

COCKBURN, A. **Writing Effective Use Cases**, The Crystal Collection for Software Professionals: Addison-Wesley Professional, 2000. 88p.

FIELDING, R. **A little REST and Relaxation**. The International Conference on Java Technology (JAZOON07), Zurique, Suíça, Junho 2007.

FIELDING, R. et al. **RFC 2616:** Hypertext Transfer Protocol -- HTTP/1.1. Internet Engineering Task Force, 1999. Disponível em http://www.w3.org/Protocols/rfc2616/rfc2616.html. Acessado em: 23 sep. 2011.

GEER, D. Are Companies Actually Using Secure Development Life Cycles? **Computer Magazine**, Volume 43, número 6, Junho 2010. p.12-16.

GLISSON, B. et al. **Secure Web Application Development and Global Regulation.** The Second International Conference on Availability, Reliability and Security, ARES 2007, 10-13 April 2007. p.681-688.

GRAHAM, D. et al. **Foundations of Software Testing**: ISTQB Certification. Cengage Learning Business Press, Nova Iorque, EUA, 2006. 382p.

HOWARD, J.; MEUNIER, P. **Using a "common language" for computer security incident information**. In "Computer Security Handbook, 4th ed.", M. Kabay e S. Bosworth, eds., Nova Iorque: Wiley. 2002. 3.4p.

HUANG, Y. et al. **Non-Detrimental Web Application Security Scanning.** Proceedings of the 15th International Symposium on Software Reliability Engineering, França, Novembro 2004. p.219-230.

KALS, S. et al. **SecuBat**: A Web Vulnerability Scanner. Proceedings of the 15th International Conference on World Wide Web, Edimburgo, Escócia, Maio 2006. p.247-256.

KIEZUM, A. et al. **Automatic creation of SQL injection and cross-site scripting attacks**. MIT Computer Science and Artificial Intelligence Laboratory technical report. Cambridge, EUA, Setembro 2008. 11p.

MCGRAW, G. et al. Interview: Software Security in the Real World**.** IEEE Pervasive Computer, **IEEE Computer**, 2010. 2p.

MCGRAW, G. et al. **BSI-MM:** The Software Security Framework. BSI-MM, 2009. Disponível em http://bsimm.com/online/. Acesso em 10 fev. 2011.

MCGRAW, G.; POTTER, B. Software Security Testing. **IEEE Security & Privacy**. Vol.2. IEEE Computer, 2004. p.32-36.

MEAD, N.; JARZOMBEK, J. Advancing Software Assurance with Public-Private Collaboration. **Computer Magazine**, Setembro 2010, vol.43, no.9, pp.21-30.

MICHAEL, C.; RADOSEVICH, W. **Black Box Security Testing Tools**. Cigital, 2009. Disponível em https://buildsecurityin.us-cert.gov/bsi/articles/tools/black-box/261-BSI.html. Acesso em 10 fev. 2011.

MITRE. **Common Weakness Enumeration 2011**. Disponível em http://cwe.mitre.org. Acesso em 23 set. 2011.

MYERS, G. **The art of software testing**. John Wiley & Sons, Inc, 2a edição, 2004. p1.

SHAHRIAR, H.; ZULKERNINE, M. **Automatic Testing of Program Security Vulnerabilities**. Proceedings of the 1st International Workshop on Test Automation, Seattle, EUA, Julho 2009. pp 550-555.

OWASP. **The OWASP Top 10 2010.** Disponível em https://www.owasp.org/index.php/Top_10_2010. Acesso em 23 set. 2011.

THOMPSON, H. Why Security Testing Is Hard. **IEEE Security and Privacy**, Julho-Agosto, 2003. p.83-86.

ZHANG, L. et al. **D-WAV**: A Web Application Vulnerabilities Detection Tool Using Characteristics of Web Forms. Fith International Conference on Software Engineering Advances, France, Agosto 2010. p.501-507.