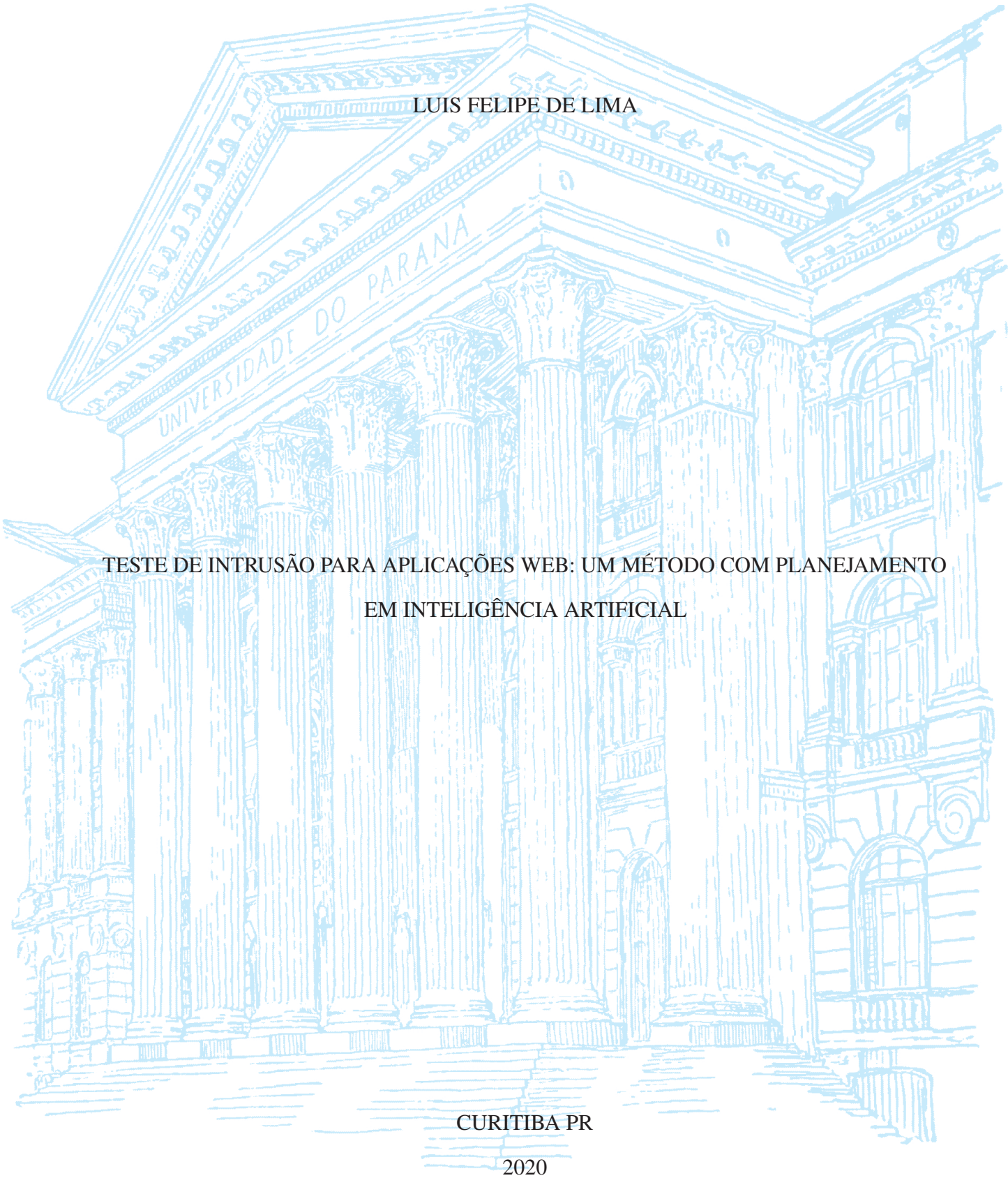
UNIVERSIDADE FEDERAL DO PARANÁ

LUIS FELIPE DE LIMA



TESTE DE INTRUSÃO PARA APLICAÇÕES WEB: UM MÉTODO COM PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

CURITIBA PR

2020

LUIS FELIPE DE LIMA

TESTE DE INTRUSÃO PARA APLICAÇÕES WEB: UM MÉTODO COM PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Informática no Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, da Universidade Federal do Paraná.

Área de concentração: *Ciência da Computação*.

Orientador: Profª. Drª. Leticia Mara Peres.

Coorientador: Prof. Dr. André Ricardo Abed Grégio.

CURITIBA PR

2020

Catalogação na Fonte: Sistema de Bibliotecas, UFPR
Biblioteca de Ciência e Tecnologia

L732t Lima, Luis Felipe de
Teste de intrusão para aplicações web: um método com planejamento em inteligência artificial [recurso eletrônico] / Luis Felipe de Lima. – Curitiba, 2020.

Dissertação - Universidade Federal do Paraná, Setor de Ciências Exatas, Programa de Pós-Graduação em Informática, 2020.

Orientadora: Leticia Mara Peres.
Coorientador: André Ricardo Abed Grégio.

1. Inteligência artificial. 2. Planejamento. 3. Software - Testes. I. Universidade Federal do Paraná. II. Peres, Leticia Mara. III. Grégio, André Ricardo Abed. IV. Título.

CDD: 006.3

Bibliotecária: Vanusa Maciel CRB- 9/1928

MINISTÉRIO DA EDUCAÇÃO
SETOR DE CIENCIAS EXATAS
UNIVERSIDADE FEDERAL DO PARANÁ
PRÓ-REITORIA DE PESQUISA E PÓS-GRADUAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO INFORMÁTICA - 40001016034P5

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em INFORMÁTICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de**LUIS FELIPE DE LIMA** intitulada: **TESTE DE INTRUSÃO PARA APLICAÇÕES WEB: UM MÉTODO COM PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL**, sob orientação da Profa. Dra. LETICIA MARA PERES, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.

A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

CURITIBA, 06 de Março de 2020.

LETICIA MARA PERES
Presidente da Banca Examinadora

ANDRÉ RICARDO ABED GRÉGIO
Coorientador (UNIVERSIDADE FEDERAL DO PARANÁ)

LUIS CARLOS ERPEN DE BONA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

ALEX MATEUS PORN
Avaliador Externo (INSTITUTO FEDERAL DO PARANÁ)

MARIA CLAUDIA FIGUEIREDO PEREIRA EMER
Avaliador Externo (UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ)

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO PARANÁ
Setor de Ciências Exatas
Coordenação de Pós-Graduação em Informática

À minha avó, Lourdes Simplício Bersi (in memoriam).

AGRADECIMENTOS

À Universidade Federal do Paraná (UFPR), pela excelência no ensino ofertado nos cursos de graduação e pós-graduação. Tenho orgulho de dizer que sou UFPR. Universidade pública, gratuita e plural que sempre defenderei.

Ao Departamento de Informática (DInf) e ao Programa de Pós-Graduação em Informática (PPGInf), pela estrutura disponibilizada para a realização deste trabalho.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela concessão da bolsa de estudos.

À minha orientadora Leticia, por todos os ensinamentos e conselhos que me fizeram crescer não apenas como pesquisador, mas também como pessoa. Agradeço pela confiança depositada em mim desde a graduação.

Aos professores do PPGInf, em especial ao meu coorientador André Grégio e ao Fabiano, pelas imprescindíveis contribuições para a elaboração deste trabalho.

Aos professores da banca examinadora, Alex, Luis Bona e Maria Claudia, pela disponibilidade em avaliar este trabalho e pelas relevantes considerações apresentadas.

Aos colegas do laboratório FAES, pelas contribuições em nossas reuniões do grupo de pesquisa, pelos momentos de convívio e café.

Aos meus familiares, que sempre se fizeram presentes me motivando para a conclusão desta etapa.

Aos meus pais, Cleide e Valdecir, por todo apoio, compreensão, paciência e incentivo durante minha vida acadêmica. Definitivamente, sem vocês eu não teria chegado até aqui.

RESUMO

Ataques a aplicações Web ocorrem com a exploração de falhas denominadas vulnerabilidades com o objetivo de obtenção de acesso à aplicação. As vulnerabilidades podem ser detectadas com uma técnica de teste de segurança chamada teste de intrusão, sendo que a execução deste teste pode requerer grande esforço dos testadores. Devido à característica sequencial presente em várias etapas que compõem um teste de intrusão, este tipo de teste vem sendo associado a problemas de planejamento em inteligência artificial (IA). Com a realização de mapeamentos sistemáticos e revisões da literatura, constatou-se que pesquisadores vêm propondo a modelagem de vulnerabilidades como problemas de planejamento em IA, com o intuito de automatizar parte do processo de teste de intrusão. No entanto, tais propostas não priorizam a modelagem da execução de ferramentas utilizadas neste tipo de teste. Esta dissertação propõe um método automatizável de teste de intrusão para aplicações Web utilizando a técnica de planejamento em IA. O método gera, em uma primeira etapa, planos de teste a partir da modelagem da execução das ferramentas de teste de intrusão como um problema de planejamento em IA. Em uma segunda etapa, os planos de teste devem ser seguidos para a execução automática destas ferramentas. A utilização do plano de teste tem como objetivo indicar ao testador as ferramentas e configurações necessárias para sua execução de acordo com o tipo de aplicação sob teste para o teste de determinada vulnerabilidade. Além disso, o método inclui uma proposta de módulo automatizável para busca de códigos de exploração de vulnerabilidades e atualização de um *framework* de teste de intrusão. Com a realização de um estudo exploratório, foram selecionadas para uso no método as ferramentas de teste de intrusão Arachni, HTCAP, Skipfish, SQLmap, Wapiti, XSSer e ZAP, além do *framework* Metasploit. Assim, a modelagem apresentada restringiu-se às vulnerabilidades injeção de SQL e *cross-site scripting* (XSS). Foi conduzido um estudo de caso a fim de se exemplificar uma aplicação do método em testes para as vulnerabilidades injeção de SQL e XSS. Durante o estudo de caso, o plano de teste mostrou-se promissor como um auxílio aos testadores na definição e execução do teste de intrusão. Ademais, o planejamento em IA mostrou-se eficaz para a modelagem do teste de intrusão e definição criteriosa das ferramentas necessárias neste tipo de teste.

Palavras-chave: Planejamento de Teste, Teste de Segurança, Detecção de Vulnerabilidades, Planejamento em Inteligência Artificial.

ABSTRACT

Web applications attacks occur with the exploitation of failures called vulnerabilities, in order to gain access to these applications. Vulnerabilities can be detected with a security testing technique called intrusion testing whose execution can take a lot of effort from testers. Due to intrusion testing sequential steps, this type of testing has been associated with artificial intelligence (AI) planning problems. With literature reviews we found proposals of vulnerabilities modeling as AI planning problems in order to automate part of the intrusion testing process. However, modeling the execution of the tools used in this type of testing was not prioritize by these proposals. This dissertation proposes an intrusion testing method for Web applications with the AI planning technique. In a first step, the method generates testing plans based on modeling the execution of the intrusion testing tools as an AI planning problem. In a second step, the testing plans must be followed for the automatic execution of these tools. Testing plans aim to indicate to the tester the tools and configurations necessary for its execution, according to the type of application under testing and vulnerability. In addition, the method includes a module that can be automated to search for exploits and update an intrusion testing framework. We conducted an exploratory study and selected the Arachni, HTCAP, Skipfish, SQLmap, Wapiti, XSSer and ZAP intrusion testing tools, and the Metasploit framework. Thus, the AI planning modeling is restricted to SQL injection and cross-site scripting (XSS) vulnerabilities. We realized a case study to exemplify an application of the method in tests for SQL injection and XSS vulnerabilities. Testing plans proved to be promising as an aid to testers in specifying and executing the intrusion testing. Also, AI planning proved to be effective in modeling the intrusion testing for defining the tools needed for this type of testing.

Keywords: *Testing Planning*, *Security Testing*, *Vulnerabilities Detection*, *Artificial Intelligence Planning*.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ACRÔNIMOS

ADL	*Action Description Language*
API	*Application Programming Interface*
AST	Aplicação Sob Teste
BAMS	*Behavioral Adversary Modeling System*
BDES	*Bottleneck Diagnosis Expert System*
BeEF	*Browser Exploitation Framework*
BIES	*Bottleneck Improvement Expert System*
BFS	*Breadth-First Search*
bWAPP	*buggy Web Application*
CI	Critério de Inclusão
CE	Critério de Exclusão
CVE	*Common Vulnerabilities and Exposures*
DInf	Departamento de Informática
DVWA	*Damn Vulnerable Web Application*
ER	Engenharia de Requisitos
EHC	*Enforced-Hill-Climbing*
FAES	Fundamentos e Aplicações em Engenharia de Software
FF	*Fast Forward*
GFC	Grafo de Fluxo de Controle
GPS	*General Problem Solver*
HTN	*Hierarchical Task Network*
HTTP	*HyperText Transfer Protocol*
HTML	*HyperText Markup Language*
IA	Inteligência Artificial
IPP	*Interference Progression Planner*
IronWASP	*IronWeb Application Advanced Security Testing Platform*
ISSAF	*Information Systems Security Assessment Framework*
MEA	*Means-Ends Analysis*
MSL	Mapeamento Sistemático da Literatura
OSSTMM	*Open Source Security Testing Methodology Manual*
OWASP	*Open Web Application Security Project*
PATHS	*Planning Assisted Tester for grapHical user interface Systems*
PDDL	*Planning Domain Definition Language*
POMDP	*Partially Observable Markov Decision Proccesses*
PPGInf	Programa de Pós-Graduação em Informática

PTES	*Penetration Testing Execution Standard*
QP	Questão de Pesquisa
SQL	*Structured Query Language*
STRIPS	*Stanford Research Institute Problem Solver*
TCP	*Transmission Control Protocol*
TLS	*Transport Layer Security*
UFPR	Universidade Federal do Paraná
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
XML	*eXtensible Markup Language*
XSS	*Cross-Site Scripting*
XVWA	*Xtreme Vulnerable Web Application*
XXE	Entidades Externas de XML
W3af	*Web Application Attack and Audit Framework*
ZAP	*Zed Attack Proxy*

SUMÁRIO

1 INTRODUÇÃO

De modo a introduzir os temas referentes a presente pesquisa, este capítulo está estruturado da seguinte forma. A Seção 1.1 apresenta uma visão geral sobre teste de intrusão para aplicações Web e planejamento em inteligência artificial (IA). Em seguida, a Seção 1.2 apresenta a motivação para o desenvolvimento desta dissertação. Adiante, a Seção 1.3 apresenta o objetivo geral e os objetivos específicos definidos. As contribuições desta dissertação são apresentadas na Seção 1.4. Finalizando o capítulo, na Seção 1.5 é apresentada a organização do trabalho.

1.1 VISÃO GERAL

A popularização das aplicações Web cresceu consideravelmente nos últimos anos. Somada à popularização, observa-se também a facilidade de acesso a este tipo de aplicação (Andrews e Whittaker, 2006). Comumente, as aplicações Web oferecem serviços que lidam com dados confidenciais e sensíveis dos usuários. A facilidade de acesso e a criticidade destes serviços são fatores que propiciam o aumento de interesse de ataques a estas aplicações. Tais ataques objetivam, por exemplo, a obtenção de acesso a dados do usuário de forma ilegal.

A comunicação entre o usuário e uma aplicação Web é estabelecida por uma rede cliente-servidor (Andrews e Whittaker, 2006). Neste tipo de rede, o cliente encaminha dados de requisição ao servidor que computa os dados recebidos e envia uma resposta contendo outros dados ao cliente. Além disso, as aplicações Web são compostas por linguagens e protocolos específicos, como Protocolo de Transferência de Hipertexto (HTTP, do inglês *HyperText Transfer Protocol*) e Linguagem de Marcação de Hipertexto (HTML, do inglês *HyperText Markup Language*). Logo, as aplicações Web requerem técnicas de teste especializadas, que atendam às especificidades de rede, linguagem e protocolos utilizados.

A exploração de aplicações Web ocorre devido a falhas denominadas vulnerabilidades. As vulnerabilidades podem ocorrer devido a erros de projeto ou implementação das aplicações (Felderer et al., 2016). Equívocos cometidos pelos desenvolvedores durante o desenvolvimento e mecanismos insuficientes de segurança podem estar associados à exploração de vulnerabilidades neste tipo de aplicação. Assim, é essencial a realização de testes para identificar a existência de vulnerabilidades para, deste modo, minimizar os riscos associados à exploração das mesmas.

Teste de software é realizado com o intuito de identificar a existência de defeitos no sistema (Myers, 1979). Geralmente, a execução de um teste de software é auxiliada por um artefato de gerenciamento chamado plano de teste de software que contém informações referentes ao teste, como recursos disponíveis e ferramentas requeridas. O plano de teste é elaborado durante a atividade de planejamento de teste de software, onde são especificados procedimentos para o teste (Sommerville, 2012). Em contextos específicos, o teste de software pode ser realizado para verificar requisitos não-funcionais do sistema, como segurança. Os testes de segurança acontecem em um ambiente controlado com o objetivo de revelar falhas exploráveis nas aplicações, sendo possível identificar se as propriedades de segurança especificadas foram implementadas corretamente (Felderer et al., 2016).

Existem diferentes técnicas de teste de segurança, como a baseada em modelos, a baseada em código, o teste de regressão e o teste de intrusão (Felderer et al., 2016). O teste de intrusão tem como objetivo a detecção de vulnerabilidades nas aplicações. Durante os testes de intrusão, o testador atua como um invasor, simulando ataques contra a aplicação (Weidman, 2014). As vulnerabilidades mais comuns em aplicações Web são listadas no OWASP Top 10

(OWASP, 2017), projeto desenvolvido e mantido por uma comunidade aberta composta por membros da área de segurança computacional. A atualização periódica do OWASP Top 10 indica o surgimento constante de novas vulnerabilidades. Consequentemente, tem-se a necessidade de novas ferramentas, técnicas e métodos de teste para detecção destas vulnerabilidades.

Os testes de intrusão são executados de acordo com metodologias que fornecem etapas sequenciais que guiam os testadores na especificação e execução dos testes. O Padrão de Execução de Teste de Intrusão (PTES, do inglês *Penetration Testing Execution Standard*) (PTES, 2017) é um exemplo de metodologia de teste de intrusão específica para aplicações Web. De acordo com etapas presentes nesta metodologia, as ferramentas de teste de intrusão são implementadas geralmente com funcionalidades de instrumentalização do teste, varredura de aplicação e exploração de vulnerabilidades. Com a execução das ferramentas de instrumentalização, obtém-se informações sobre a aplicação sob teste, como o identificador de sessão associado à autenticação da mesma. As ferramentas para varredura tem como objetivo identificar a existência de vulnerabilidades na aplicação sob teste. Já as ferramentas de exploração objetivam que o testador obtenha acesso à aplicação sob teste a partir das possíveis vulnerabilidades encontradas.

Dadas certas características de uma aplicação Web, como autenticação, nota-se que a execução de um teste de intrusão neste tipo de aplicação pode requerer expertise dos testadores. Desta forma, um teste de intrusão pode demandar grande esforço (Andrews e Whittaker, 2006). Assim, percebe-se a importância de se associar técnicas que auxiliem os testadores na definição, especificação, planejamento e execução dos testes de intrusão. A característica de execução sequencial das ferramentas de teste de intrusão indica a possibilidade de sua representação com técnicas de modelagem.

Planejamento em inteligência artificial (IA) é uma técnica que permite a modelagem de problemas por meio da execução sequencial de um conjunto de ações para atingir determinado objetivo (Russell e Norvig, 2016). Desta forma, o encadeamento das ferramentas que compõem o fluxo de um teste de intrusão pode ser diretamente associado a um problema de planejamento em IA. Um problema de planejamento em IA modelado em alguma linguagem formal é utilizado como entrada de uma ferramenta de planejamento, também chamada de planejador. A execução de um planejador gera um conjunto de ações que leva do estado inicial ao final do problema, ou seja, gera um dos possíveis planos para a resolução do problema.

As linguagens formais de planejamento em IA, em sua maioria, contêm sintaxe de fácil aprendizado e consequentemente, de fácil modelagem. Como exemplo, tem-se a Linguagem de Definição de Domínio de Planejamento (PDDL, do inglês *Planning Domain Definition Language*) (McDermott et al., 1998). A PDDL contém funcionalidades referentes a operações com valorações numéricas, permitindo associar métricas à geração dos planos, ocasionando uma representação mais efetiva do problema de planejamento em IA. Além destas vantagens associadas às linguagens, a utilização do planejamento em IA é facilitada pelo ferramental de código aberto disponibilizado pela comunidade de inteligência artificial.

Na literatura, a técnica de planejamento em IA vem sendo aplicada a teste de intrusão com diferentes abordagens. Nas propostas de Bozic e Wotawa (2017) e Bozic e Wotawa (2018) os autores propõem a modelagem em PDDL de vulnerabilidades como problemas de planejamento em IA. Outra abordagem é identificada nos trabalhos de Obes et al. (2013) e Shmaryahu et al. (2018), em que os autores realizam a modelagem em PDDL de sequências de comandos chamados *exploits*, executados na aplicação com o intuito de explorar determinada vulnerabilidade. Esta dissertação se difere destas abordagens pois propõe a modelagem em PDDL da execução de ferramentas de teste de intrusão como um problema de planejamento em IA.

Neste contexto, esta dissertação tem como proposta um método de teste de intrusão para aplicações Web com planejamento em IA. O método proposto associa os planos de teste obtidos

a partir da modelagem com planejamento em IA com a execução automática de ferramentas de teste de intrusão. O método contém atividades de planejamento do teste e execução do teste. Durante a atividade de planejamento do teste, a execução das ferramentas de teste de intrusão é modelada na linguagem PDDL, gerando planos de teste. Estes são obtidos com a execução de um planejador com uma métrica que prioriza a ferramenta de varredura que detecta mais vulnerabilidades. Além disso, são utilizados como critérios para a geração do plano o tipo de aplicação sob teste e a vulnerabilidade que se deseja testar.

Desta forma, o plano de teste contém ações que indicam as configurações necessárias de cada ferramenta para testes de uma determinada vulnerabilidade em um determinado tipo de aplicação. Assim, os planos de teste são utilizados como auxílio aos testadores durante a definição da atividade de execução do teste. A execução do teste no método proposto é realizada automaticamente, com a integração de ferramentas para instrumentalização do teste, varredura de aplicação e exploração de vulnerabilidades. O método contém também uma proposta de módulo automatizável para busca de códigos de exploração de vulnerabilidades, executados em casos onde não há ferramenta de exploração disponível para a vulnerabilidade identificada na varredura.

1.2 MOTIVAÇÃO

Como aspectos que motivaram o desenvolvimento desta dissertação, destacam-se: (i) a criticidade dos serviços oferecidos pelas aplicações Web; (ii) a dificuldade de padronizar a execução das ferramentas de testes de intrusão, dadas características pertinentes às aplicações Web; (iii) a grande quantidade de recursos de tempo e pessoal que a execução manual de um teste de intrusão pode requerer; (iv) a facilidade de modelagem de problemas com a linguagem PDDL; (v) a funcionalidade de inclusão de variáveis numéricas da linguagem PDDL, permitindo a definição de métricas na geração dos planos de teste; (vi) as ferramentas de teste de intrusão e planejamento em IA de código aberto disponíveis; e (vii) o fato das propostas identificadas na literatura não realizarem a modelagem da execução de ferramentas de teste de intrusão utilizando planejamento em IA.

1.3 OBJETIVOS

Diante das diferentes possibilidades de configurações de ferramentas de teste de intrusão de acordo com a aplicação que se deseja testar e da possibilidade da execução sequencial destas ferramentas, são propostos os objetivos a seguir.

- **Objetivo geral**:
  - propor um método automatizável de teste de intrusão para aplicações Web a partir de planos de teste obtidos com a modelagem com planejamento em IA.
- **Objetivos específicos**:
  - modelar a execução de ferramentas de teste de intrusão como um problema de planejamento em IA;
  - gerar os planos de teste com uma ferramenta de planejamento em IA;
  - aplicar o método com a execução automática de testes de intrusão a partir dos planos de teste gerados.

1.4 CONTRIBUIÇÃO

Esta dissertação contribui para o planejamento e execução de testes de intrusão em aplicações Web. Como contribuições desta dissertação, destacam-se: (i) revisão da literatura sobre a utilização de planejamento em IA em engenharia de requisitos, design de software, teste de software e teste de intrusão; (ii) análise de ferramentas de teste de intrusão e aplicações sob teste; (iii) definição da atividade de planejamento de teste no contexto de teste de intrusão para aplicações Web; (iv) geração de planos de teste de intrusão a partir da modelagem com a técnica de planejamento em IA; e (v) definição de um método automatizável de execução de teste de intrusão.

1.5 ORGANIZAÇÃO DO TRABALHO

Além do capítulo introdutório, esta dissertação está organizada nos seguintes capítulos. No Capítulo 2 é apresentada a fundamentação teórica com definições e conceitos referentes a teste de software e teste de intrusão, sendo abordadas vulnerabilidades em aplicações Web, metodologia PTES e ferramentas desta técnica de teste. São também apresentadas as definições de planejamento em IA, onde são abordadas a linguagem PDDL, um exemplo de problema modelado nesta linguagem e planejadores. Ainda neste capítulo são apresentados trabalhos relacionados a planejamento em IA em teste de intrusão.

No Capítulo 3 é apresentado um estudo exploratório com ferramentas de teste de intrusão, onde são realizadas análises destas ferramentas e aplicações sob teste.

No Capítulo 4 é apresentado o método de teste de intrusão a partir de planos de teste gerados com a modelagem em PDDL. São abordadas as atividades de planejamento do teste e execução do teste que compõem o método proposto.

No Capítulo 5 é apresentada uma aplicação do método proposto, sendo apresentados a metodologia e os resultados de um estudo de caso elaborado para esta finalidade.

Finalizando esta dissertação, o Capítulo 6 apresenta considerações finais, limitações identificadas durante seu desenvolvimento, suas contribuições e sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a fundamentação teórica e os trabalhos relacionados ao desenvolvimento desta dissertação. A Seção 2.1 apresenta conceitos referentes a teste de software, seguido de definições das técnicas e fases de teste de software. A Seção 2.2 apresenta os conceitos de segurança de software e teste de intrusão. Na Subseção 2.2.1 são apresentadas definições de vulnerabilidades em aplicações Web. Finalizando a Seção 2.2, a Subseção 2.2.2 descreve a metodologia Padrão de Execução de Teste de Intrusão (PTES). A Seção 2.3 apresenta definições das ferramentas utilizadas na execução de um teste de intrusão. A Seção 2.4 apresenta a definição da técnica de planejamento em inteligência artificial (IA). Na Subseção 2.4.1 é descrita a Linguagem de Definição de Domínio de Planejamento (PDDL). Em seguida, na Subseção 2.4.2 é apresentado um exemplo de modelagem e resolução de um problema de planejamento na linguagem PDDL. Finalizando a Seção 2.4, a Subseção 2.4.3 apresenta as ferramentas de planejamento em IA. A Seção 2.5 apresenta trabalhos relacionados a planejamento em IA aplicado a teste de intrusão. Por fim, a Seção 2.6 apresenta considerações sobre o capítulo.

2.1 TESTE DE SOFTWARE

De acordo com Myers (1979), **teste de software** é definido como o processo de executar um programa em um ambiente controlado com a intenção de revelar defeitos[1]. A execução de um programa contendo defeitos pode ocasionar um resultado diferente do resultado esperado, ou seja, uma **falha**. Assim, os testes são fundamentais para a garantia de qualidade de software, contribuindo para determinar se o mesmo está de acordo com a especificação e podendo oferecer informações para as atividades de depuração[2] e manutenção[3] (Pressman, 2011).

Um teste consiste na análise da execução de um determinado caso de teste. **Caso de teste** é um par ordenado formado por um dado de entrada, chamado **dado de teste**, e a respectiva **saída esperada** de sua execução. Um programa é considerado correto quando o seu comportamento está de acordo com o comportamento esperado para todos os dados de teste (Delamaro et al., 2007).

Conforme Delamaro et al. (2007), um programa T contém um domínio de entrada D. Diz-se que o programa T é correto para uma especificação S se $S(d) = T(d)$, para qualquer dado de teste $d \in D$. Ou seja, T é correto se a saída obtida com a execução de todos os dados de teste está de acordo com a saída esperada. Neste contexto, um caso de teste é definido pelo par ordenado $(d, S(d))$, sendo $d \in D$ e $S(d)$ sua respectiva saída esperada.

- **Técnicas de teste de software**

Idealmente, o teste de um programa T deveria abranger todos os dados de teste d pertencentes ao domínio de entrada D. No entanto, isto é considerado impraticável devido às restrições de recursos e ao tamanho de D. Deste modo, costuma-se estabelecer critérios para seleção de casos de teste, na tentativa de diminuir os custos associados à atividade de teste.

Critério de teste é um predicado que define quais propriedades de um programa T devem ser executadas visando obter um teste sistemático (Delamaro et al., 2007). Em outras

[1] **Defeito** é uma linha de código, bloco ou conjunto de dados incorretos (Wazlawick, 2013).

[2] **Depuração** é o processo de identificar a causa do defeito revelado pelo teste (Wazlawick, 2013).

[3] **Manutenção** é o processo de adaptação, otimização e correção de defeitos de um software já desenvolvido (Wazlawick, 2013).

palavras, um critério de teste indica os casos de teste mais adequados para a execução do teste, de modo a aumentar as chances da revelação de defeitos em T. Os critérios de teste são definidos, geralmente, a partir das técnicas funcional, estrutural e baseada em erros.

A técnica de **teste funcional** (Myers, 1979), conhecida como teste de caixa-preta, estabelece os critérios de teste a partir das funções de especificação do software. Nesta técnica há pouca preocupação com relação à implementação e estrutura lógica interna do software (Pressman, 2011). Assim, é fundamental que a documentação do software seja bem elaborada, abrangendo todas as funcionalidades do sistema e requisitos do usuário. Particionamento em classes de equivalências e análise do valor limite (Delamaro et al., 2007) são exemplos de critérios desta técnica.

A técnica de **teste estrutural** (Myers, 1979), ou teste de caixa-branca, baseia-se no conhecimento da estrutura interna do software para a definição dos critérios de teste. Em geral, os critérios desta técnica utilizam o grafo de fluxo de controle (GFC) (Delamaro et al., 2007) como representação do programa em teste T. O GFC é um grafo orientado $G = (V, E, s)$, onde V é o conjunto de vértices, E é o conjunto de arcos e $s \in V$ é o vértice de entrada. Cada vértice $v \in V$ representa um bloco indivisível de comandos do programa T, ou seja, quando um comando de um bloco é executado os demais são executados sequencialmente. Cada arco de E representa um possível desvio entre os blocos de V. O GFC contém um único vértice de entrada $s \in V$ e um único vértice de saída. Um **caminho** é uma sequência finita de vértices $(v_1, v_2, ..., v_k), k \geq 2$, tal que existe um arco de v_i para v_{i+1}, $i = 1, 2, ..., k - 1$.

Os critérios de teste estrutural são classificados, geralmente, de acordo com o fluxo de controle (como desvios condicionais) e o fluxo de dados (como definições e referências de variáveis) do programa em teste T. Para o critério de fluxo de controle, deve-se escolher o componente do GFC que será executado durante o teste. Dentre estes critérios, destacam-se o critério **Todos-Nós**, que requer que todos os vértices do GFC sejam executados ao menos uma vez; o critério **Todos-Arcos**, que exige que todos os arcos do grafo devem ser executados; e o critério **Todos-Caminhos**, que requer que todos os caminhos do GFC sejam executados.

A técnica de **teste baseada em erros** estabelece critérios de teste por meio de informações sobre os erros mais frequentes no processo de desenvolvimento de software. Desta forma, esta técnica tem como objetivo determinar maneiras de detectar a ocorrência dos erros que o programador possa cometer durante o desenvolvimento. Análise de mutantes é um dos critérios comumente utilizados nesta técnica (Delamaro et al., 2007).

- **Fases de teste de software**

As técnicas de teste são utilizadas ao longo do desenvolvimento de software de acordo com uma estratégia de teste pré-definida, contendo fases de teste de unidade, integração, sistema e validação (Pressman, 2011; Wazlawick, 2013). O **teste de unidade** procura identificar erros de implementação e lógica nos componentes individuais do software. Estes componentes (unidades) podem ser métodos, procedimentos, classes ou conjunto de funções de tamanho pequeno. Cada unidade é testada individualmente pelo próprio programador.

O **teste de integração** é realizado quando as unidades estão finalizadas, testadas individualmente e serão integradas para gerar uma versão do sistema. Estes testes são realizados pelos programadores e equipe de teste, com o objetivo principal de identificar erros associados à interface do sistema. **O teste de sistema** é realizado pela equipe de teste com o sistema totalmente integrado, com o objetivo de identificar erros em funções e características que não estejam de acordo com sua especificação. Além disso, verifica se todas as unidades do sistema combinam-se adequadamente. Já o **teste de aceitação** é realizado por usuários finais com a

interface final do sistema. Desta maneira, o objetivo do teste de aceitação é realizar a validação do sistema quanto aos requisitos do usuário.

Uma estratégia de teste geralmente utilizada prevê atividades de teste associadas a cada fase de teste e a diferentes fases de desenvolvimento de software. Tais atividades referem-se ao planejamento de teste e execução de testes. A atividade de **planejamento de teste de software** consiste em especificar padrões e procedimentos para o teste, estabelecer *checklists* para orientar os testadores e definir um plano de teste de software (Sommerville, 2012).

Um **plano de teste de software** é um artefato de gerenciamento destinado aos testadores. Além das fases envolvidas no teste, o plano de teste deve especificar os sistemas a serem testados, bem como estabelecer cronograma, alocação de recursos e restrições que podem afetar a realização dos testes. Ademais, o plano de teste deve estabelecer as ferramentas requeridas para a realização dos testes (Sommerville, 2012). Após a atividade de planejamento de teste ser concluída, inicia-se a atividade de **execução de teste de software**, realizada pelos testadores com o auxílio do plano de teste definido.

Esta estratégia envolvendo planejamento de teste e execução de teste é descrita no Modelo V (Pressman, 2011), que se trata de um modelo de desenvolvimento de software apresentado na Figura 2.1. O lado esquerdo do modelo mostra as fases de desenvolvimento onde os testes são planejados. Uma vez realizada a geração de código, inicia-se a execução de testes para verificação e validação dos requisitos, modelos e implementações do sistema. Estes testes ocorrem sequencialmente como descrito no lado direito do modelo. As linhas tracejadas representam o planejamento do teste e as linhas contínuas a execução do teste.

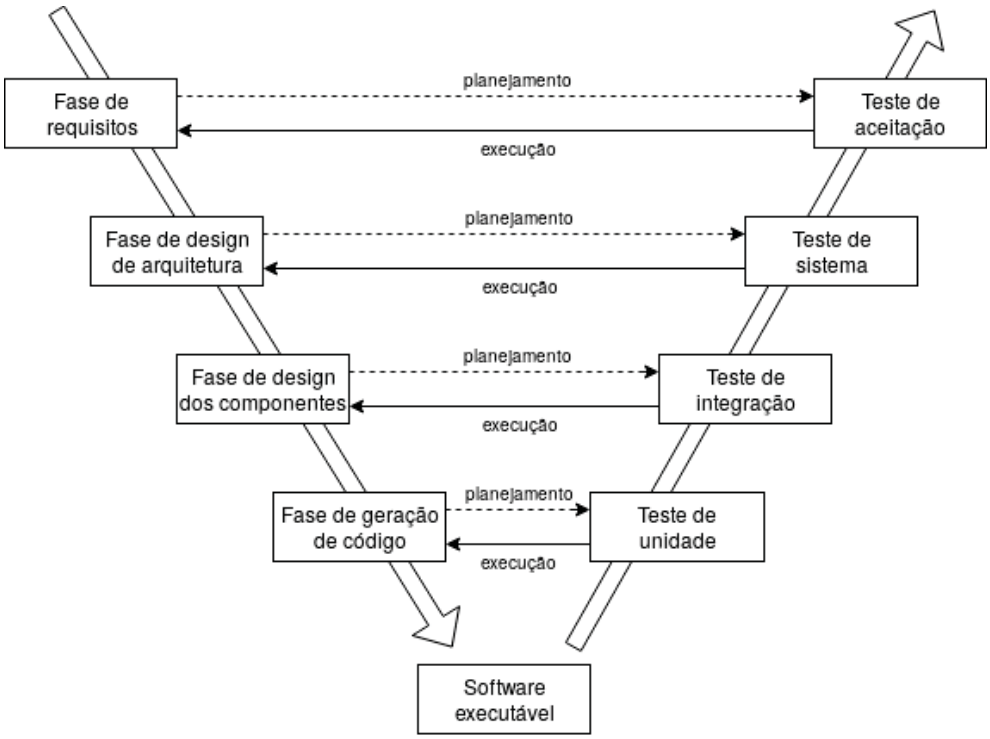


Figura 2.1: Atividades de teste do Modelo V. Fonte: Adaptado de Pressman (2011).

O Modelo V indica que os testes de aceitação devem ser planejados durante a fase de levantamento de requisitos. O planejamento do teste de sistema deve ser realizado durante a fase de design de arquitetura, quando os requisitos são organizados em unidades funcionais coesas definindo como as diferentes partes arquiteturais do sistema irão se conectar. Já os testes de aceitação devem ser planejados durante a fase de design de componentes, quando

são aprofundadas as descrições de todas as partes do sistema e são tomadas as decisões de implementação. Por fim, o modelo indica que o planejamento dos testes de unidade deve ocorrer durante a fase de geração do código.

2.2 TESTE DE INTRUSÃO

Em contextos específicos, os testes podem ser executados para verificar requisitos não-funcionais do sistema, como desempenho, usabilidade e segurança. **Segurança** é uma característica de qualidade de software definida pela ISO/IEC 25010:2011 (2011) como o grau em que funções e dados de um software são protegidos de acesso não autorizado e o grau em que são disponibilizados para acesso autorizado. É subdividida em cinco subcaracterísticas, apresentadas a seguir.

- **Autenticidade**: grau em que a identidade de uma pessoa ou recurso é efetivamente aquela que se diz ser;
- **Confidencialidade**: grau em que um software assegura que os dados sejam acessíveis somente para aqueles autorizados a ter acesso;
- **Integridade**: grau em que um software impede o acesso por pessoas ou sistemas não autorizados;
- **Não repúdio**: grau em que um software permite constatar que ações ou acessos foram efetivamente realizados;
- **Rastreabilidade de uso**: grau em que as ações realizadas por uma pessoa ou sistema podem ser rastreadas de forma a comprovar que foram efetivamente realizadas por estes.

Teste de segurança é definido como o processo de execução do sistema com o objetivo de realizar sua verificação e validação quanto aos requisitos de segurança relacionados à autenticidade, confidencialidade, integridade, não repúdio e rastreabilidade de uso (Felderer et al., 2016). Por meio dos testes de segurança é possível identificar se as propriedades de segurança especificadas são implementadas de maneira correta. Assim, os testes de segurança são realizados durante a fase de teste de sistema, quando uma versão totalmente integrada do sistema está disponível.

Teste baseado em modelos, teste baseado em código, teste de regressão e teste de intrusão são exemplos de técnicas de teste de segurança (Felderer et al., 2016). Esta dissertação de mestrado tem como foco o **teste de intrusão**, técnica de teste de segurança cuja finalidade é detectar a existência de vulnerabilidades no sistema (Weidman, 2014). **Vulnerabilidade** é uma falha que permite que um invasor obtenha acesso ilegal ao sistema (Felderer et al., 2016). As vulnerabilidades podem estar associadas a falhas em nível de projeto ou a erros de implementação, e sua existência pode ocasionar danos a diferentes partes interessadas, como proprietário, usuários e outras entidades que dependem do sistema.

Conforme apontado por Weidman (2014), o teste de intrusão é uma simulação de um ataque ao sistema com o objetivo de identificar e explorar vulnerabilidades. Assim, a execução de testes de intrusão ocorre em um ambiente controlado, com o testador agindo como um invasor. A exploração de vulnerabilidades identificadas permite que o testador obtenha e possivelmente mantenha acesso ao sistema. Com isso, o invasor pode obter informações confidenciais do usuário e realizar ataques subsequentes, por exemplo. O teste de intrusão é realizado com o sistema em execução, geralmente de forma remota e sem conhecimento sobre o funcionamento interno do sistema. Desta forma, pode ser associado a uma técnica de teste funcional.

2.2.1 Vulnerabilidades em aplicações Web

No contexto Web, os alvos de um teste de intrusão são aplicações que comunicam-se com o usuário/testador de acordo com um protocolo de rede, como Protocolo de Controle de Transmissão (TCP, do inglês *Transmission Control Protocol*) (Andrews e Whittaker, 2006). Para esta dissertação, a aplicação-alvo de um teste é denominada **aplicação sob teste (AST)**.

As vulnerabilidades mais comuns em aplicações Web são apresentadas no OWASP Top 10 (OWASP, 2017), projeto idealizado e mantido pela comunidade Projeto Aberto de Segurança em Aplicações Web (OWASP, do inglês *Open Web Application Security Project*), composta por membros da área de segurança computacional. A comunidade OWASP tem como objetivo possibilitar que organizações concebam, desenvolvam, adquiram, operem e mantenham aplicações confiáveis. A lista de vulnerabilidades do OWASP Top 10 é atualizada de acordo com o aumento da exploração de determinada vulnerabilidade ou com o surgimento de novas vulnerabilidades.

A edição mais recente do OWASP Top 10 divulgada refere-se ao ano de 2017 (OWASP, 2017), apresentada a seguir.

V1 **Injeção de SQL** (do inglês, *Structured Query Language injection*): ocorre quando dados não confiáveis são enviados como parte de um comando ou consulta. Possibilita que o invasor execute comandos ou acesse dados não autorizados;

V2 **Quebra de autenticação** (do inglês, *broken authentication*): ocorre quando funcionalidades da aplicação relacionadas à autenticação são implementadas de maneira incorreta. Possibilita que o invasor comprometa senhas ou explore outras falhas para assumir temporariamente ou permanentemente identidades de outros usuários;

V3 **Exposição de dados sensíveis** (do inglês, *sensitive data exposure*): ocorre quando dados confidenciais não são protegidos adequadamente pela aplicação. Possibilita que o invasor obtenha ou modifique estes dados;

V4 **Entidades externas de XML** (XXE, do inglês *Extensible Markup Language external entities*): ocorre quando processadores são configurados de forma incorreta. Possibilita que o invasor explore o processador e obtenha informações;

V5 **Quebra de controle de acesso** (do inglês, *broken access control*): ocorre quando as restrições sobre o que o usuário autenticado pode fazer não são aplicados corretamente. Possibilita que o invasor acesse dados ou funcionalidades não autorizados;

V6 **Configurações de segurança incorretas** (do inglês, *security misconfiguration*): ocorre quando a configuração padrão de segurança do sistema é realizada de forma inadequada. Possibilita que o invasor obtenha acesso ou conhecimento sobre o sistema;

V7 ***Cross-site scripting* (XSS)**: ocorre quando um aplicativo inclui dados não confiáveis em uma página Web sem a devida validação. Possibilita que o invasor execute *scripts* no navegador da vítima;

V8 **Desserialização insegura** (do inglês, *insecure deserialization*): ocorre por meio da alteração de objetos serializados, geralmente associados à execução de código remoto. Possibilita que o invasor realize ataques de repetição, injeção e escalonamento de privilégios;

V9 **Utilização de componentes vulneráveis** (do inglês, *using components with known vulnerabilities*): ocorre quando um componente (biblioteca, estrutura ou módulos do software) vulnerável é explorado. Possibilita que o invasor obtenha dados do sistema;

V10 **Registo e monitorização insuficiente** (do inglês, *insufficient logging & monitoring*): ocorre quando o sistema é ineficiente quanto à obtenção de *logs*, monitoramento e resposta a incidentes. Possibilita que o invasor continue atacando o sistema de forma persistente, adulterando ou extraindo dados.

2.2.2 Metodologia PTES

Os testes de segurança são realizados de acordo com etapas sequenciais fornecidas por metodologias. Cada etapa de uma metodologia contém atividades para o planejamento, execução e posterior análise dos resultados obtidos nos testes. Desta forma, a utilização de uma metodologia auxilia os testadores na execução de testes padronizados. *Framework* de Avaliação de Segurança de Sistemas de Informação[4] (ISSAF, do inglês *Information Systems Security Assessment Framework*), Guia de Teste OWASP[5] (do inglês, *OWASP Testing Guide*) e Manual de Metodologia de Teste de Segurança de Código Aberto[6] (OSSTMM, do inglês *Open Source Security Testing Methodology Manual*) são exemplos destas metodologias.

Padrão de Execução de Teste de Intrusão (PTES, do inglês *Penetration Testing Execution Standard*) (PTES, 2017) é um exemplo de metodologia específica para teste de intrusão. As primeiras etapas desta metodologia especificam atividades para obtenção de informações da AST. Estas informações são utilizadas na especificação dos testes, executados posteriormente com o objetivo de detectar e explorar vulnerabilidades na AST. As vulnerabilidades detectadas proporcionam que o testador explore a AST, obtendo mais informações que permitem a realização de ataques subsequentes. Além disso, a metodologia PTES fornece atividades para a análise de vulnerabilidades, culminando em um relatório sobre o teste realizado.

A metodologia PTES é composta por sete etapas (PTES, 2017), realizadas conforme fluxo apresentado na Figura 2.2. As etapas da metodologia PTES são descritas a seguir.

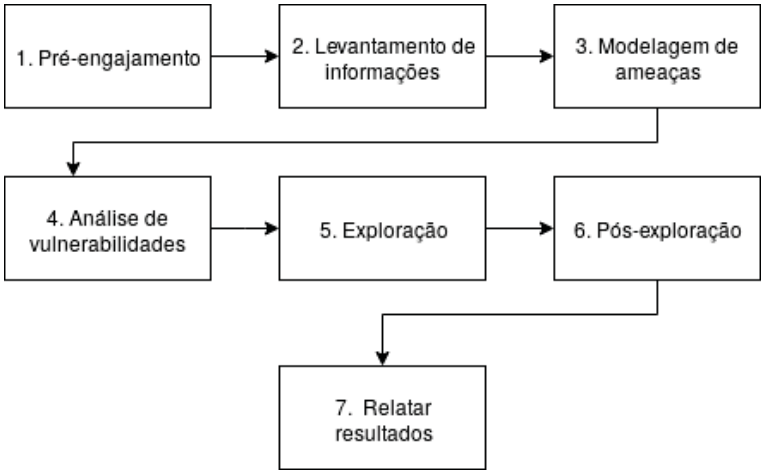


Figura 2.2: Fluxo das etapas da metodologia PTES. Fonte: Adaptado de PTES (2017).

[4]Disponível em: https://untrustednetwork.net/files/issaf0.2.1.pdf
[5]Disponível em: https://www.owasp.org/images/1/19/OTGv4.pdf
[6]Disponível em: https://www.isecom.org/OSSTMM.3.pdf

1. **Pré-engajamento** (do inglês, *pre-engagement interactions*): etapa para definir o escopo e objetivo do teste de intrusão;

2. **Levantamento de informações** (do inglês, *intelligence gathering*): etapa para coletar informações da AST, como usuário e senha, por exemplo;

3. **Modelagem de ameaças** (do inglês, *threat modeling*): etapa para desenvolver estratégias para minimizar riscos associados a presença de vulnerabilidades na AST;

4. **Análise de vulnerabilidades** (do inglês, *vulnerability analysis*): etapa para identificar vulnerabilidades que podem ser exploradas pelo testador na AST;

5. **Exploração** (do inglês, *exploitation*): etapa para o testador estabelecer acesso à AST a partir de vulnerabilidades existentes;

6. **Pós-exploração** (do inglês, *post exploitation*): etapa para manter acesso à AST e esconder possíveis evidências da invasão;

7. **Relatar resultados** (do inglês, *reporting*): etapa para produzir um relatório com os resultados obtidos nos testes.

2.3 FERRAMENTAS DE TESTE DE INTRUSÃO

A partir da classificação de ferramentas de teste de intrusão realizada no trabalho de mestrado "Web (Eternamente) Revisitada: Análise de Vulnerabilidades Web e de Ferramentas de Código Aberto para Exploração" (Neto, 2019), são apresentadas nesta seção definições de tais ferramentas contendo a formalização de alguns de seus aspectos, como conjuntos de vulnerabilidades, entradas e saídas.

Para esta dissertação, as aplicações sob teste (ASTs) são aplicações Web, acessadas por meio de Localizadores Uniformes de Recursos (URL, do inglês *Uniform Resource Locator*) (Comer, 2016). Uma AST contém um conjunto $Pg_{ast} = \{pg_{ast_1}, pg_{ast_2}, ..., pg_{ast_k} | k \in \mathbb{N}\}$ de k páginas acessadas por URLs distintas e pode conter um conjunto $V_{ast} = \{v_{ast_1}, v_{ast_2}, ..., v_{ast_n} | n \in \mathbb{N}\}$ de n vulnerabilidades. As URLs de uma AST são utilizadas como entradas pelas ferramentas de teste de intrusão e deste modo podem representar dados de teste. Caso necessite de autenticação, uma AST possui usuário e senha que estabelecem um respectivo identificador de sessão do tipo *string* associado à autenticação.

As ferramentas para teste de intrusão são desenvolvidas com funcionalidades distintas de acordo com as etapas da metodologia selecionada para a definição dos testes. Assim, estas ferramentas são utilizadas de maneira encadeada, a fim de atingir o objetivo específico de cada etapa da metodologia. De modo geral, a execução de um teste de intrusão requer ferramentas com três funcionalidades: levantamento de informações da AST, detecção de vulnerabilidades e exploração das vulnerabilidades identificadas. Estas ferramentas são associadas, respectivamente, às etapas "2. Levantamento de informações", "4. Análise de vulnerabilidades" e "5. Exploração" da metodologia PTES, apresentadas em 2.2.2. O fluxo de execução das ferramentas para teste de intrusão é apresentado na Figura 2.3.

Conforme apresentado na Figura 2.3, a presença de autenticação na AST é a característica que determina o fluxo de ferramentas que será executado no teste. Esta característica da AST indica a necessidade de utilização de uma configuração específica de cada ferramenta. Uma configuração é caracterizada por um conjunto de parâmetros da ferramenta para a execução do teste da AST. As entradas das ferramentas indicadas por I1, I2 e I3 representam configurações

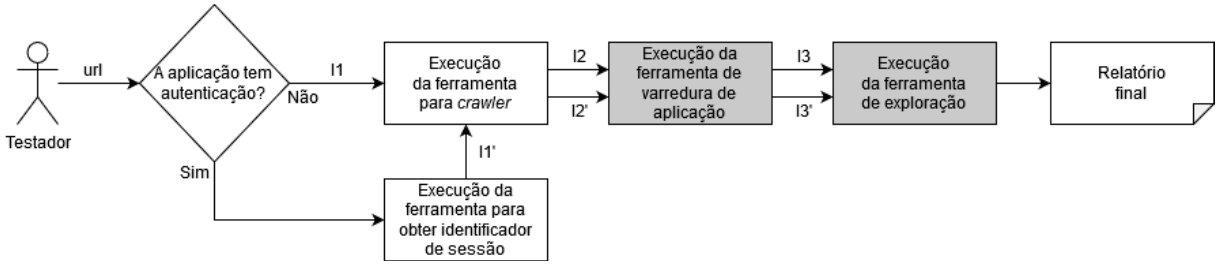


Figura 2.3: Fluxo de execução de ferramentas de teste de intrusão. Fonte: O autor (2019).

para ASTs que não requerem autenticação. Já as entradas indicadas por I1', I2' e I3' representam as configurações para AST com autenticação. Tais entradas são descritas adiante.

O levantamento de informações da AST ocorre com a execução da ferramenta para *crawler*, cujo objetivo é identificar as URLs das páginas da AST que serão submetidas posteriormente à varredura, e da ferramenta para obtenção do identificador de sessão de AST que contém autenticação. As execuções destas ferramentas, indicadas pelas caixas em branco na Figura 2.3, proporcionam a instrumentalização do teste, pois fornecem informações que serão utilizadas pelas demais ferramentas do fluxo de execução. Já as execuções das ferramentas de varredura de aplicação e exploração, destacadas em cinza na Figura 2.3, detectam a presença de vulnerabilidades na AST e exploram as vulnerabilidades identificadas. Ou seja, representam a execução do teste propriamente dito.

A seguir, são apresentadas definições das ferramentas para *crawler*, varredura de aplicação e exploração, elaboradas para esta dissertação de mestrado. Além destas definições, são realizadas também descrições das entradas de cada ferramenta, apresentadas na Figura 2.3.

- **Ferramenta para *crawler***

O teste de intrusão inicia-se com o testador informando a URL da AST que é utilizada como entrada do *crawler*. Se a AST requerer autenticação, é necessário definir previamente um usuário e uma senha. O processo de autenticação gera um identificador de sessão associado ao usuário e senha criados. O identificador de sessão é obtido com uma ferramenta específica e encaminhado juntamente com a URL da AST para a execução do *crawler*. Portanto, estas ferramentas proporcionam a instrumentalização do teste, pois são obtidas informações que possibilitam a definição e execução do teste com as ferramentas subsequentes.

A presença de autenticação na AST indica a possibilidade de duas entradas distintas para a execução do *crawler*, conforme apresentado na Figura 2.3: I1, para AST sem autenticação e I1', para AST com autenticação. A entrada I1 é composta apenas pela URL da AST, já I1' é composta da URL e do identificador de sessão associado à autenticação realizada. Consequentemente, o *crawler* é executado com configurações distintas, caracterizadas por parâmetros específicos de acordo com a entrada utilizada. Como resultado da execução do *crawler*, obtém-se um conjunto $Pg_c \subseteq Pg_{ast} = \{pg_{c_1}, pg_{c_2}, ..., pg_{c_m} | m \in \mathbb{N}\}$ de m páginas da AST. O testador indica a profundidade de execução do *crawler*, determinando a quantidade de páginas obtidas.

- **Ferramenta de varredura de aplicação**

As ferramentas de varredura de aplicação, também conhecidas como *scanners*, são ferramentas de teste de intrusão executadas com o objetivo de identificar a existência de um conjunto $V_v = \{v_{v_1}, v_{v_2}, ..., v_{v_t} | t \in \mathbb{N}\}$ de t vulnerabilidades na AST. Para isso, utiliza como dado de teste o conjunto Pg_c de páginas identificadas pelo *crawler*. Além deste conjunto de páginas,

são informados a URL da AST e o identificador de sessão, quando autenticação é requerida pela AST.

Desta maneira, as execuções da ferramenta de varredura são realizadas com configurações distintas de acordo com o tipo de AST que está sendo testada. Conforme indicado na Figura 2.3, I2 indica a utilização do conjunto Pg_c de páginas como dado de teste de AST que não requer autenticação. Já I2' refere-se à AST com autenticação, ou seja, é composto pelo identificador de sessão e pelo dado de teste Pg_c.

A execução da ferramenta de varredura resulta em um relatório com um conjunto $Pg_v \subseteq Pg_c = \{pg_{v_1}, pg_{v_2}, ..., pg_{v_m} | v_1, v_2, .., v_m \in V_v, m \in \mathbb{N}\}$ de m páginas da AST com a indicação de vulnerabilidades identificadas na varredura. Algumas destas ferramentas possuem a funcionalidade de *crawler*, acionada por meio de um parâmetro específico. Informações referentes à AST, como URL e identificador de sessão, e ao relatório de saída, como nome, formato e diretório de destino, também requerem a utilização de parâmetros específicos.

- **Ferramenta de exploração**

O teste de intrusão é finalizado com a execução de uma ferramenta de exploração. As ferramentas de exploração, como o próprio nome indica, são ferramentas de teste de intrusão que objetivam a exploração de vulnerabilidades presentes em páginas da AST identificadas previamente com a varredura, permitindo que o testador obtenha acesso à AST. Estas ferramentas exploram um conjunto $V_e = \{v_{e_1}, v_{e_2}, ..., v_{e_q} | q \in \mathbb{N}\}$ de q vulnerabilidades. Além das informações da AST, as ferramentas de exploração utilizam como dado de teste o conjunto $Pg_e \subseteq Pg_v = \{pg_{e_1}, pg_{e_2}, ..., pg_{e_z} | e_1, e_2, ..., e_z \in V_e, z \in \mathbb{N}\}$ de z páginas da AST.

De forma análoga ao *crawler* e à ferramenta de varredura, as ferramentas de exploração são executadas com configurações distintas de acordo com a AST a ser testada. Estas configurações requerem dois tipos de entrada, apresentadas na Figura 2.3: I3, para AST sem autenticação e I3', para AST com autenticação. Desta forma, Pg_e compõe o dado de teste em I3 enquanto Pg_e e o identificador de sessão formam I3'. Como saída da execução da exploração, obtém-se um relatório final do teste de intrusão, contendo informações sobre a exploração realizada. Para a execução da ferramenta de exploração, parâmetros específicos são utilizados para as informações da AST e do relatório de saída.

- ***Framework* para teste de intrusão**

Os testes de intrusão também podem ser realizados com o auxílio de *frameworks*, estruturados com a integração de um conjunto de ferramentas. Desta maneira, os *frameworks* podem conter diferentes funcionalidades, como obtenção de informações, varredura de aplicação e exploração de vulnerabilidades. Uma abordagem adotada pelos *frameworks* é a execução de *exploits* (Weidman, 2014), que se tratam de sequências de comandos que realizam comportamento imprevisto na AST por meio da exploração de vulnerabilidades existentes. A execução bem sucedida de um *exploit* indica evidências de que a aplicação é vulnerável.

As ferramentas de exploração executam os testes utilizando uma base local de *exploits*. Em casos onde o objetivo do teste seja explorar uma vulnerabilidade que não esteja contida no conjunto V_e de vulnerabilidades da ferramenta de exploração, os *frameworks* de teste de intrusão podem ser executados como uma alternativa. Os *frameworks* de teste de intrusão permitem a inclusão de *exploits* para exploração de vulnerabilidades específicas. Estes *exploits* adicionais podem ser elaborados pelo testador ou encontrados em bases de dados disponibilizadas pela comunidade de segurança computacional, como a Exploit-DB (2019). Estas bases de dados podem utilizar identificadores padronizados de vulnerabilidades, como os apresentados pela lista Vulnerabilidades e Exposições Comuns (CVE, do inglês *Common Vulnerabilities and Exposures*) (CVE, 2019).

2.4 PLANEJAMENTO EM INTELIGÊNCIA ARTIFICIAL

Planejamento em inteligência artificial (IA) é definido como a elaboração automática de uma sequência de ações para atingir o objetivo de um problema (Russell e Norvig, 2016). Ou seja, dado um problema com uma determinada situação inicial, um objetivo específico e um conjunto de ações, o propósito do planejamento em IA é encontrar uma sequência de ações que atinja o objetivo a partir da situação inicial.

Conforme Russell e Norvig (2016), um problema de planejamento em IA é descrito como uma quádrupla (***P, I, O, A***), onde:

- $\boldsymbol{P} = \{p_1, p_2, ..., p_m\} | m \in \mathbb{N}\}$ é um conjunto de m predicados;
- $\boldsymbol{I} \in S$ é o estado inicial do problema contido no conjunto S de estados;
- $\boldsymbol{O} \in S$ é o estado final do problema contido no conjunto S de estados; e
- $\boldsymbol{A} = \{a_1, a_2, ...a_n\} | n \in \mathbb{N}\}$ é um conjunto de n ações.

Os estados do conjunto S de um problema de planejamento em IA são especificados por predicados de P. Os predicados são instanciados por objetos e, ao especificar determinado estado, são considerados verdadeiros neste estado. Além disso, os predicados são utilizados para a definição de todas as ações de A. Uma ação a é descrita em termos de pré-condições e efeitos (funções $pre(a)$ e $post(a)$, respectivamente). Se todas as funções $pre(a)$ de uma ação a são verdadeiras em um estado $S_1 \in S$ então a é executada e ocorre a transição para um outro estado $S_2 \in S$, ou seja, $S_1 \xrightarrow{a} S_2$. Depois da transição, os efeitos $post(a)$ da ação a são automaticamente considerados como válidos no estado S_2.

Um **plano** é um conjunto de k ações que ao serem executadas levam do estado inicial $I \in S$ ao estado final $O \in S$ de um determinado problema, ou seja $I \xrightarrow{a_1} S_1 \xrightarrow{a_2} S_2 ... \xrightarrow{a_k} O$. Em outras palavras, plano é uma possível solução para um problema de planejamento em IA. Diferentes planos podem ser gerados de acordo com o algoritmo utilizado para a resolução do problema (Russell e Norvig, 2016).

Uma ação é dita determinística quando sua execução em um estado leva a um único outro estado. Esta característica das ações permite que um problema de planejamento em IA seja representado por um grafo orientado $G = (V, E)$, denominado **grafo de planejamento**. Neste grafo, cada vértice $v \in V$ representa um estado do problema de planejamento em IA e cada arco $e \in E$, rotulado com uma ação, representa uma transição entre os estados. Um plano é representado por um caminho em G (Ghallab et al., 2004).

2.4.1 Linguagem de Definição de Domínio de Planejamento (PDDL)

Proposto por Newell et al. (1959), o sistema de busca Solucionador de Problemas Gerais (GPS, do inglês *General Problem Solver*) foi a base para o surgimento do Solucionador de Problemas do Instituto de Pesquisa de Stanford (STRIPS, do inglês *Stanford Research Institute Problem Solver*) (Fikes e Nilsson, 1971). STRIPS, nome dado à linguagem e ao sistema de resolução, influenciou a definição de linguagens modernas como Linguagem de Descrição de Ações (ADL, do inglês *Action Description Language*) (Pednault, 1989) e Linguagem de Definição de Domínio de Planejamento (PDDL, do inglês *Planning Domain Definition Language*). (McDermott et al., 1998). Desde seu surgimento, a PDDL vem sendo utilizada como linguagem-padrão em competições de planejamento em IA (Russell e Norvig, 2016).

A representação de problemas de planejamento em IA utilizando PDDL é realizada com a definição de dois arquivos: um referente ao domínio e outro ao problema (McDermott et al., 1998). O arquivo de **domínio** (`dominio.pddl`) contém o conhecimento do problema a ser resolvido. Este conhecimento é descrito pelo conjunto P de predicados e pelo conjunto A de ações do problema. Além do nome do domínio e de requisitos da linguagem, este arquivo também pode conter a definição opcional de tipos, constantes e funções.

As funções representam variáveis numéricas associadas às ações, funcionalidade que difere a PDDL de outras linguagens de planejamento em IA. As funções podem representar restrições numéricas (em $pre(a)$ e O) ou efeitos numéricos (em $post(a)$). As restrições numéricas podem requerer comparações entre valores de variáveis e constantes. Com os efeitos numéricos é possível realizar operações aritméticas em variáveis e constantes (Hoffmann, 2003).

O arquivo do **problema** (`problema.pddl`) representa uma instância do problema a ser resolvido, caracterizada pelos estados inicial e final do problema. Desta forma, estão presentes neste arquivo o nome do problema, o domínio no qual o problema está associado, os objetos do problema e a descrição dos estados I e O. Métricas, obtidas a partir de funções declaradas no arquivo de domínio, podem ser utilizadas opcionalmente. Diferentes arquivos de problema podem ser associados a um mesmo arquivo de domínio, justificando a representação do problema de planejamento em IA em dois arquivos distintos.

2.4.2 Exemplo de um problema em PDDL

Esta seção apresenta um exemplo de modelagem do problema caminho mínimo como um problema de planejamento em IA na linguagem PDDL. A escolha deste problema para a modelagem se deve pois sua definição em PDDL necessita de variáveis numéricas, permitindo exemplificar a definição e utilização desta funcionalidade, que se trata de um diferencial desta linguagem.

- **Definição do problema de planejamento**

Para este exemplo, o problema caminho mínimo é caracterizado por um conjunto de cidades conectadas entre si com determinadas distâncias. A resolução do problema consiste em descobrir o caminho de menor distância entre a cidade de origem e a cidade de destino.

O grafo direcionado ponderado apresentado na Figura 2.4 representa o problema que será modelado em PDDL neste exemplo. Este exemplo contém um conjunto de onze cidades `c1`, `c2`, `c3`, ..., `c11` conectadas, sendo `c1` a cidade de origem e `c11` a cidade de destino. No grafo, as cidades são representadas pelos vértices, enquanto os arcos representam a conexão existente entre duas cidades. A distância existente entre as cidades é representada pelo peso associado aos arcos.

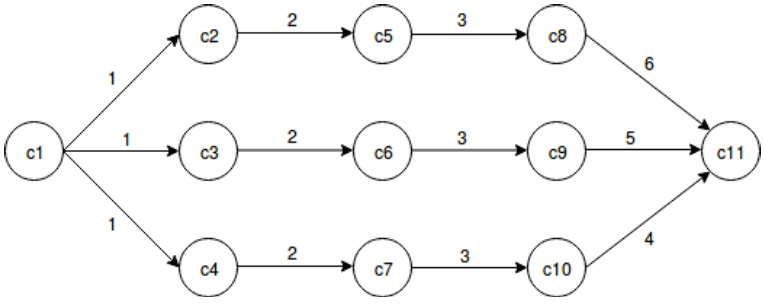


Figura 2.4: Grafo do problema caminho mínimo. Fonte: O autor (2019).

Em termos de problema de planejamento em IA, as cidades são representadas por estados, enquanto o deslocamento entre duas cidades representa a execução de uma ação. Já as distâncias existentes entre as cidades são representadas por variáveis numéricas.

- **Modelagem do problema de planejamento**

A modelagem na linguagem PDDL, realizada após a definição do problema de planejamento em IA, resulta nos arquivos `dominio_caminho_minimo.pddl` e `problema_caminho_minimo.pddl`, apresentados e descritos a seguir.

- **Modelagem do domínio em arquivo PDDL**

O domínio referente ao problema caminho mínimo, especificado no arquivo `dominio_caminho_minimo.pddl`[7], apresentado a seguir. A declaração do nome do domínio é realizada por meio da palavra *domain*, reservada da linguagem PDDL. Os demais componentes do domínio são definidos em seções de requisitos, tipos, predicados, funções e ações, descritas adiante.

```
;arquivo dominio_caminho_minimo.pddl
(define (domain caminho_minimo)
  (:requirements :typing :fluents)
  (:types cidade)

  (:predicates
    (em ?atual - cidade)
    (conexao ?atual ?fim - cidade))

  (:functions
    (distancia ?atual - cidade ?fim - cidade)
    (distancia_total))

  (:action ir
    :parameters (?atual - cidade ?fim - cidade)
    :precondition (and (em ?atual)
                       (conexao ?atual ?fim))
    :effect (and (em ?fim)
                 (not (em ?atual))
                 (increase (distancia_total)
                           (distancia ?atual ?fim)))
  )
)
```

A seguir, são detalhadas as seções do arquivo de domínio, exemplificadas com as informações do arquivo `dominio_caminho_minimo.pddl`.

- **Requisitos**: são declarados na seção *:requirements* representando as características da linguagem PDDL presentes no arquivo. As declarações são realizadas por meio de palavras reservadas da linguagem. No exemplo, são utilizados os requisitos *:typing*, que indica que cada objeto do domínio deve ser declarado com um respectivo tipo e *:fluents*, que estabelece que variáveis numéricas são declaradas no arquivo.

[7] A descrição de comentários em arquivos PDDL inicia-se com o marcador ";".

- **Tipos**: a declaração de tipos é realizada na seção *:types*, como no exemplo em que o tipo `cidade` é declarado. Os tipos declarados devem ser utilizados para a especificação dos objetos do arquivo.

- **Predicados**: a declaração de predicados é realizada na seção *:predicates*. Os predicados representam valores *booleanos* e são compostos por um nome e objetos, que devem ser tipados de acordo com os tipos declarados no domínio. No exemplo, os predicados declarados são compostos por objetos do tipo `cidade`. O predicado `em` refere-se à cidade `atual` em que o problema se encontra. Já o predicado `conexao` estabelece que existe uma conexão entre as cidades `atual` e `fim`.

- **Funções**: a utilização de variáveis numéricas, estabelecida pelo requisito *:fluents*, acontece na seção de funções (*:functions*). As funções são declaradas com a mesma sintaxe dos predicados, porém diferem-se pois representam valores numéricos. No exemplo, a função `distancia` representa a distância existente entre as cidades `atual` e `fim` e a função `distancia_total` representa a distância do caminho percorrido.

- **Ações**: as ações de um domínio são declaradas, individualmente, em seções *:action*. Todas as declarações realizadas anteriormente no domínio (tipos, predicados e funções) são utilizados na especificação das ações. Uma ação é composta por nome, lista de parâmetros (especificado por *:parameters*), pré-condições e efeitos (especificados por *:precondition* e *:effect*, respectivamente). As pré-condições e efeitos são constituídos por uma lista de predicados, agrupados pela palavra reservada *and*. É possível também realizar a negação de um predicado com a palavra *not*.

  O exemplo apresentado contém a ação `ir`, que representa a transição entre as cidades `atual` e `fim`, passadas como parâmetros. Como pré-condições, deve-se estar na cidade `atual` que deve estar conectada com a cidade `fim`. Como efeitos da execução da ação, passa-se a estar na cidade `fim` e, consequentemente, deixando de estar na cidade `atual`.

  As funções especificadas no domínio também são utilizadas para a definição das ações. O valor numérico contido nas funções pode ser alterado por operações aritméticas ou por palavras reservadas da linguagem PDDL para incremento ou decremento. No exemplo, a palavra *increase* é utilizada para incrementar o valor da função `distancia_total` com o valor da função `distancia` a cada execução da ação `ir`. Assim, ao término de todas as execuções da ação, a função `distancia_total` contém o valor total da distância percorrida até a cidade de destino.

- **Modelagem do problema em arquivo PDDL**

O arquivo do problema é modelado com o intuito de descrever uma instância do domínio `caminho_minimo`. Inicialmente, é estabelecida a associação com o domínio que o problema deverá instanciar, por meio da palavra reservada *domain*. O restante da definição do problema é realizada por meio das seções de objetos, estado inicial, estado final e métrica. O arquivo `problema_caminho_minimo.pddl`, que contém a modelagem do problema de acordo com as informações apresentadas na Figura 2.4, é apresentado a seguir.

```
;arquivo problema_caminho_minimo.pddl
(define (problem caminho_minimo)
  (:domain caminho_minimo)
```

```
  (:objects c1 c2 c3 c4 c5 c6
            c7 c8 c9 c10 c11 - cidade)
  (:init

    (em c1)

    (conexao c1 c2)  (conexao c1 c3)
    (conexao c1 c4)  (conexao c2 c5)
    (conexao c3 c6)  (conexao c4 c7)
    (conexao c5 c8)  (conexao c6 c9)
    (conexao c7 c10) (conexao c8 c11)
    (conexao c9 c11) (conexao c10 c11)

    (= (distancia c1 c2) 1)  (= (distancia c1 c3) 1)
    (= (distancia c1 c4) 1)  (= (distancia c2 c5) 2)
    (= (distancia c3 c6) 2)  (= (distancia c4 c7) 2)
    (= (distancia c5 c8) 3)  (= (distancia c6 c9) 3)
    (= (distancia c7 c10) 3) (= (distancia c8 c11) 6)
    (= (distancia c9 c11) 5) (= (distancia c10 c11) 4)

    (= (distancia_total) 0))

  (:goal (and (em c11)))

  (:metric minimize (distancia_total))
)
```

A descrição das seções do arquivo do problema, exemplificadas com as informações do arquivo `problema_caminho_minimo.pddl`, é apresentada a seguir.

- **Objetos**: na seção *:objects* são declarados os objetos do problema, que devem ser tipados de acordo com os tipos declarados no arquivo de domínio. No exemplo, são declarados os objetos `c1`, `c2`, `c3`, ..., `c11` do tipo `cidade`, que representam as cidades especificadas na Figura 2.4.

- **Estado inicial**: a seção *:init* representa o estado inicial do problema, onde predicados e funções são instanciados com suas valorações iniciais. No exemplo, a cidade `c1` é definida como a cidade inicial do problema. São realizadas as indicações das conexões e distâncias existentes entre as cidades de acordo com o exemplo e indicada a valoração inicial da função `distancia_total`.

- **Estado final**: o estado final é composto por valorações de predicados e funções que representam o objetivo do problema. É definido no arquivo de problema na seção *:goal* e representado no exemplo pela cidade de destino `c11`.

- **Métrica**: a seção de métrica *(:metric)* é utilizada para indicar ao planejador uma função que deve ter seu valor associado minimizado ou maximizado. No exemplo, a palavra reservada *minimize* indica que o valor da função `distancia_total` deve ser minimizada pelo planejador.

Após a modelagem dos arquivos de domínio e problema em PDDL, é possível realizar a associação dos componentes contidos nestes arquivos com as definições de planejamento em IA apresentadas nesta seção. Assim, é obtida a descrição do problema caminho mínimo em termos de predicados, estados inicial e final e ações, como indicado pela quádrupla (*P, I, O, A*) a seguir.

- $P = \{em, conexao\}$,
- $I = \{(em\ c1)\}$,
- $O = \{(em\ c11)\}$ e
- $A = \{ir\}$.

- **Geração do plano**

A Figura 2.5 representa o grafo do problema do exemplo com o menor caminho entre as cidades `c1` e `c11` destacado pelos vértices em cinza. Este caminho é composto por cinco cidades (`c1`, `c4`, `c7`, `c10` e `c11`), percorridas com a execução de quatro ações (`ir(c1, c4)`, `ir(c4, c7)`, `ir(c7, c10)` e `ir(c10, c11)`). O caminho de menor distância entre as cidades de origem e destino apresenta distância `10`.

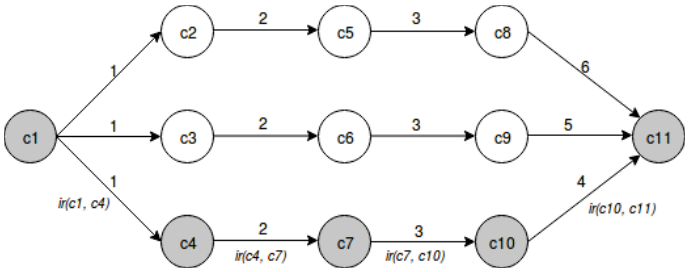


Figura 2.5: Grafo com um plano para o problema caminho mínimo. Fonte: O autor (2019).

A geração do plano ocorre com a execução de uma ferramenta de planejamento, também chamada de planejador, apresentada na seção seguinte. Para este exemplo, os planos foram gerados com o planejador Metric-FF (Hoffmann, 2003) tendo como entrada os arquivos `dominio_caminho_minimo.pddl` e `problema_caminho_minimo.pddl`. O parâmetro "`-s 3`" foi utilizado na execução do Metric-FF, indicando o algoritmo A* (Russell e Norvig, 2016) como a heurística de busca adotada pelo planejador.

O excerto da saída do Metric-FF apresentado a seguir contém as ações do plano para o problema caminho mínimo, bem como o custo associado ao plano. Como esperado, o planejador retorna o plano formado pela execução das quatros ações anteriormente citadas com custo `10`, valoração contida na função `distancia_total` ao final da execução destas ações.

```
step    0: IR C1 C4
        1: IR C4 C7
        2: IR C7 C10
        3: IR C10 C11
plan cost: 10.000000
```

Utilizando as definições de planejamento em IA, o plano gerado neste exemplo pode ser representado por $c1 \xrightarrow{ir} c4 \xrightarrow{ir} c7 \xrightarrow{ir} c10 \xrightarrow{ir} c11$.

2.4.3 Planejadores

Ferramentas de planejamento em IA, ou **planejadores**, recebem como entrada um problema de planejamento em IA descrito em alguma linguagem formal. A resolução do problema acontece de acordo com a execução de alguma heurística de busca no grafo de planejamento construído pelo planejador. O planejador tem como saída um dos possíveis planos do problema, representado por um caminho no grafo gerado.

Os planejadores são implementados de acordo com especificidades de cada linguagem de planejamento. São exemplos o planejador ABSTRIPS (Sacerdoti, 1974), para problemas na linguagem STRIPS, e UCPOP (Penberthy et al., 1992), para problemas em ADL. Para esta dissertação, foi selecionado o planejador Metric-FF (Hoffmann, 2003), que recebe como entrada problemas modelados em PDDL. O Metric-FF foi desenvolvido como um aprimoramento do planejador *Fast Forward* (FF) (Hoffmann e Nebel, 2001), diferenciando-se de seu antecessor pela capacidade de resolver problemas contendo variáveis numéricas. A Figura 2.6 apresenta o fluxo de geração de um plano com o Metric-FF.

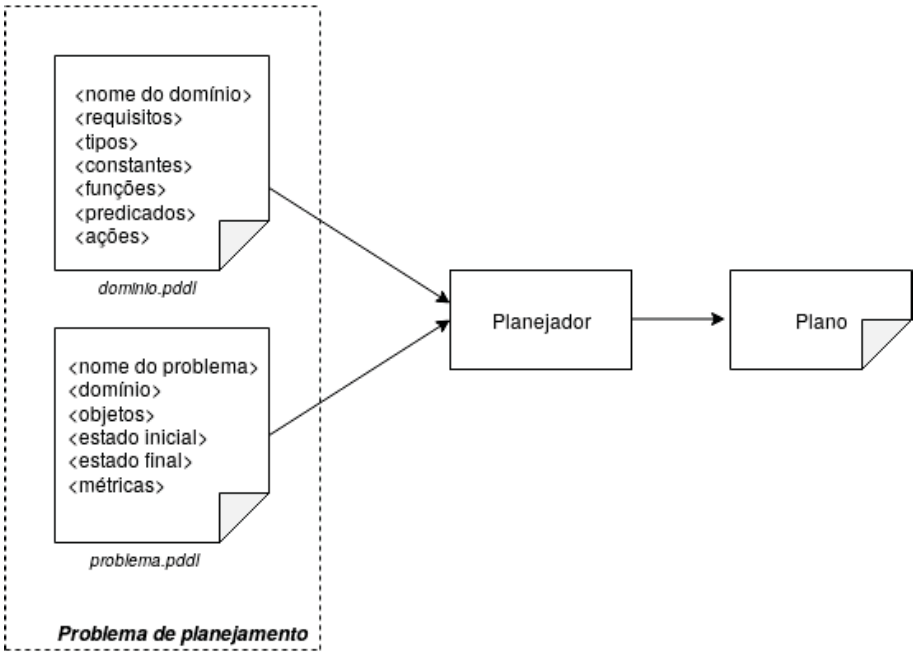


Figura 2.6: Geração de um plano com a linguagem PDDL. Fonte: O autor (2019).

O planejador Metric-FF gera um dos possíveis planos a partir da modelagem de um problema de planejamento em IA na linguagem PDDL. Desta forma, utiliza como entrada os arquivos de domínio e problema. Um exemplo de execução do planejador é dado por `./ff -o <dominio.pddl> -f <problema.pddl> -s <int>`. Os parâmetros "`-o`" e "`-f`" referem-se aos arquivos de entrada e o parâmetro "`-s`", seguido de `0` <= int <= `5`, estabelece o algoritmo de busca que será utilizado pelo planejador no grafo de planejamento para a geração do plano.

2.5 TRABALHOS RELACIONADOS

O Apêndice A apresenta os protocolos elaborados e os resultados obtidos em dois mapeamentos sistemáticos da literatura (MSLs), conduzidos a fim de se conhecer o estado da arte da utilização do planejamento em IA na engenharia de software. Os MSLs realizados baseiam-se

na metodologia proposta por Petersen et al. (2015). A realização destes MSLs tem como objetivo investigar a aplicação do planejamento em IA em diferentes fases de desenvolvimento de software. O mapeamento MSL-ReqDes, apresentado em A.1, refere-se a planejamento em IA aplicado às fases engenharia de requisitos e design. Já o mapeamento MSL-Teste, descrito em A.2, busca identificar publicações com planejamento em IA aplicado à fase de teste de software.

Nesta seção são apresentadas publicações que aplicam a técnica de planejamento em IA em teste de intrusão. A análise destas publicações objetiva identificar abordagens existentes na literatura referentes a estes tópicos e, consequentemente, apontar as diferenças do uso do planejamento em IA nesta dissertação. As abordagens identificadas referem-se à detecção de vulnerabilidades e execução de *exploits* modelados em PDDL como problemas de planejamento em IA. Além destas abordagens, adiante são apresentados alguns *surveys* sobre o uso do planejamento em IA no teste de intrusão.

- **Detecção de vulnerabilidades**

Bozic e Wotawa (2014) iniciam sua pesquisa sobre teste de intrusão com uma proposta de modelagem de especificações das vulnerabilidades injeção de SQL e XSS. Estas especificações são descritas em termos de objetivos (por exemplo, explorar a aplicação sob teste (AST)), pré-condições (por exemplo, ser possível enviar dados para a AST), ações (por exemplo, estabelecer conexão com a AST e enviar dados maliciosos) e pós-condições (por exemplo, obter informações da AST). Estas especificações são descritas como diagramas de estados UML, processo que pode ser automatizado com ferramentas específicas. Os autores utilizam como ASTs do estudo de caso cinco aplicações vulneráveis, ou seja, implementadas com vulnerabilidades pré-determinadas. Os testes realizados pelo autores mostram que a modelagem proposta é capaz de identificar a presença das vulnerabilidades nas ASTs. Os autores usam esta modelagem de vulnerabilidades como base para suas propostas com planejamento em IA.

Wotawa e Bozic (2014) propõem uma modelagem de teste de intrusão utilizando a linguagem PDDL. De acordo com as características da linguagem PDDL, o problema de planejamento em IA definido consiste em arquivos de problema e domínio. No arquivo do problema são especificados estado inicial, que contém informações como tipo de vulnerabilidade e informações da AST (URL e autenticação), e estado final, que especifica o que é esperado da AST caso uma vulnerabilidade seja detectada. Já o arquivo de domínio contém a descrição das ações, que representam possíveis ações do invasor, como enviar uma requisição para a AST, por exemplo. Os planos gerados são casos de teste abstratos, ou seja, representam sequências de ações que indicam a existência de vulnerabilidades na AST. Os autores direcionam a modelagem para a identificação das vulnerabilidades injeção de SQL e XSS. A representação em PDDL destas vulnerabilidades deriva da modelagem a partir de diagramas de estado UML, proposta pelos autores no trabalho anterior de 2014. Visando a automatização da execução do teste, os autores propõem o algoritmo PLAN4SEC que inicializa o planejador Metric-FF com os arquivos em PDDL definidos. Após isso, este algoritmo mapeia as ações contidas nos planos gerados pelo planejador para funções Java, que são executadas na AST. Resultados preliminares obtidos em um estudo de caso empírico indicam que a abordagem pode ser utilizada na prática para a detecção de vulnerabilidades.

Como forma de continuidade dos trabalhos anteriores, Bozic e Wotawa (2015) propõem a ferramenta PURITY, que integra uma segunda versão do algoritmo PLAN4SEC com o planejador Metric-FF. A ferramenta proposta tem como objetivo principal a detecção das vulnerabilidades injeção de SQL e XSS em aplicações Web. Além da URL da AST, dos arquivos em PDDL e da vulnerabilidade que se deseja testar, a ferramenta PURITY utiliza vetores de ataque como parte da entrada. Os vetores de ataque são arquivos de texto com códigos em JavaScript representando

as vulnerabilidades injeção de SQL e XSS. Devido a estes vetores de ataque, os planos gerados pelo planejador são considerados pelos autores como casos de teste concretos. A execução dos planos com os vetores de ataque na AST difere esta abordagem do trabalho anterior dos autores. A ferramenta contém uma interface gráfica que disponibiliza formas de execução manual, que requer o testador preenchendo todos os campos da interface, e completamente automática.

Bozic e Wotawa (2017) propõem uma extensão da modelagem em PDDL apresentada em Bozic e Wotawa (2015). Nesta versão da modelagem, os autores incluem ações na especificação do domínio que representam o Protocolo de Segurança da Camada de Transporte (TLS, do inglês *Transport Layer Security Protocol*) (Dierks, 2008). A seguir é apresentado um excerto do arquivo de domínio com duas ações para exemplificar a modelagem realizada pelos autores. A ação `Start` representa o início do teste, onde a URL passada como parâmetro indica a existência da AST. A execução desta ação indica que a AST foi inicializada. A ação `SendReq` representa o envio de requisições, tendo como pré-condição a inicialização da AST na ação `Start`.

```
(:action Start
  :parameters(?x - active ?url - address ?lo - status-lo)
  :precondition (and (inInitial ?x)
                     (not (Empty ?url)))

  :effect (and (inAddressed ?x)
               (not (inInitial ?x))
               (Logged yes))
)

(:action SendReq
  :parameters(?x - active ?lo - status-lo ?se - status-se
              ?si - status-si ?lo - status-lo)
  :precondition (and (inAddressed ?x)
                     (Logged yes))

  :effect (and (inSentReq ?x)
               (not (inAddressed ?x))
               (assign (sent ?se) 1)
               (statusinit two)))
)
```

Em Bozic e Wotawa (2018), trabalho identificado no mapeamento sistemático apresentado em A.2, os autores sintetizam seus trabalhos anteriores como um *framework* composto da ferramenta PURITY, do planejador JavaGP, de uma ferramenta de *crawler* e de vetores de ataque das vulnerabilidades injeção de SQL e XSS. A aplicabilidade do *framework* é demonstrada em estudo de caso tendo como AST uma aplicação Web.

- **Execução de *exploits***

A utilização do planejamento em IA para a modelagem de *exploits* é introduzida por Boddy et al. (2005). Os autores propõem o Sistema de Modelagem Adversária Comportamental (BAMS, do inglês *Behavioral Adversary Modeling System*), que seleciona os atributos e objetivos de um invasor em potencial e gera ataques que este pode realizar contra a AST. Os autores integram ao sistema BAMS o planejador Metric-FF, que obtém um plano composto de uma sequência

de *exploits* que representa um possível ataque contra o sistema. Para se obter um plano com custo mínimo, os autores utilizam a funcionalidade de métricas disponibilizada pela linguagem PDDL. Assim, o plano obtido representa o ataque com o menor risco de sucesso. O estudo de caso conduzido exemplifica um ataque contra uma aplicação Web de gestão de documentos. Um excerto da modelagem dos *exploits*, realizada na linguagem PDDL, é apresentado a seguir. A ação representa um *exploit* que modifica a lista de controle de acesso de um documento ao adicionar acesso de leitura para um determinado grupo.

```
(:action DMS_ADD_GROUP_ALLOW
  :parameters (?admin - c_human ?chost - c_host ?shost - c_host
               ?doc - c_file ?gid - c_gid)
  :precondition (and (pmode free)
                     (nes_admin_connected ?chost ?shost)
                     (at_host ?admin ?chost)
                     (insider ?admin))

  :effect (and (dmsacl_read ?doc ?gid))
)
```

Sarraute et al. (2011) apresentam uma modelagem de *exploits* em planejamento em IA. Os autores utilizam a funcionalidade de inclusão de variáveis numéricas da linguagem PDDL na modelagem. Estas variáveis representam o custo das ações em termos de tempo de execução. Os autores indicam que as variáveis numéricas podem ser utilizadas em termos de tráfego de rede gerado ou associando uma probabilidade de sucesso de execução das ações, por exemplo. Para a geração dos planos, são utilizados os planejadores Metric-FF e SGPlan. O experimento realizado em uma rede com 1000 *hosts* mostra que a abordagem é eficiente em redes de tamanho médio. A seguir é apresentado um exemplo da modelagem de um *exploit* em PDDL apresentado pelos autores. Este *exploit* tem como objetivo instalar um agente previamente instalado no *host* de origem `s` no *host* alvo `t`. Para que isso ocorra, o *host* alvo deve ter sistema operacional, arquitetura e serviços específicos. Além disso, deve existir conexão TCP entre os *hosts* de origem e alvo. Como efeito da ação, a função `time`, que representa o tempo de execução do *exploit*, é incrementada. Posteriormente, esta função é minimizada pelo planejador, gerando o plano que representa a sequência de *exploits* com menor tempo de execução.

```
(:action IBM_Tivoli_Storage_Manager_Client_Exploit
  :parameters (?s - host ?t - host)
  :precondition (and (compromised ?s)
                     (has_OS ?t Windows)
                     (has_OS_edition ?t Professional)
                     (has_OS_servicepack ?t Sp2)
                     (has_OS_version ?t WinXp)
                     (has_architecture ?t I386))
                     (has_service ?t mil-2045-47001)
                     (TCP_connectivity ?s ?t port1581))

  :effect(and (installed_agent ?t high_privileges)
              (increase (time) 4))
)
```

Como forma de continuidade do trabalho de Sarraute et al. (2011) e motivados pela dificuldade de se escolher manualmente os *exploits* adequados, Obes et al. (2013) propõem a integração de um planejador com um *framework* de teste de intrusão. A escolha de um *framework* se deve por este conter uma base de dados própria de *exploits* reais. Os autores integram o *framework* Core Impact e os planejadores Metric-FF e SGPlan por meio de um módulo chamado *PlannerRunner*. As informações da AST, da rede e dos *exploits* são mapeados de forma parcialmente automatizada para uma representação em PDDL. O plano gerado, de forma análoga ao trabalho anterior dos autores, representa a sequência de *exploits* com menor tempo de execução. O plano é então encaminhado ao *framework* Core Impact que executa os *exploits* na AST. A principal vantagem da proposta dos autores é o ganho de desempenho na execução do teste, visto que o conhecimento prévio dos *exploits* a serem executados impede que o *framework* realize uma busca em toda sua base de dados. A seguir é apresentado um excerto do plano gerado pelo planejador e executado pelo *framework*. O plano é obtido em um cenário cujo objetivo é comprometer o *host* alvo `10.0.5.12`. No excerto, são apresentados dois dos cinco *exploits* que compõem o plano, indicados nas linhas `3` e `17`.

```
0: Mark_as_compromised localagent localhost
1: IP_connect localhost 10.0.1.1
2: TCP_connect localhost 10.0.1.1 port80
3: Phpmyadmin Server_databases Remote Code Execution
localhost 10.0.1.1
4: Mark_as_compromised 10.0.1.1 high_privileges

(...)

14: Mark_as_compromised 10.0.4.2 high_privileges
15: IP_connect 10.0.4.2 10.0.5.12
16: TCP_connect 10.0.4.2 10.0.5.12 port445
17: Novell Client NetIdentity Agent Buffer Overflow
10.0.4.2 10.0.5.12
18: Mark_as_compromised 10.0.5.12 high_privileges
```

Shmaryahu et al. (2018) propõem a modelagem do teste de intrusão utilizando planejamento contingente (do inglês, *contingent planning*), modelo qualitativo em que ações são distribuídas de acordo com uma probabilidade. Os autores elaboram a proposta a partir de estudos com planejamento em IA clássico e processos de decisão de Markov parcialmente observáveis (POMDPs, do inglês *partially observable Markov decision proccesses*) (Sondik, 1978). O planejamento em IA clássico é geralmente utilizado em propostas onde estrutura e configuração da rede são conhecidas pelo testador. Já POMDPs, que se trata de um modelo probabilístico, é utilizado em propostas onde estrutura e configurações são inicialmente desconhecidas pelo testador. A principal motivação dos autores é associar a facilidade e escalabilidade para obtenção de modelos em planejamento em IA clássico com o conhecimento parcial característico de modelos em POMDPs. A modelagem é realizada com padrão de sintaxe da linguagem PDDL específico para problemas de planejamento contingente (Albore et al., 2009). A seguir é apresentado um exemplo de modelagem de um *exploit* como ação do problema de planejamento. São utilizados como parâmetros os *hosts* de origem e destino e informações de sistema operacional, software e vulnerabilidade. A ação é executada com o objetivo de obtenção de controle do *host* alvo `t` a partir da existência da vulnerabilidade `v`.

```
(:action exploit
  :parameters (?s - host ?t - host ?o - os ?sw - sw ?v - vuln)
  :precondition (and (hacl ?s ?t)
                     (controlling ?s)
                     (not (controlling ?t))
                     (HostOS ?t ?o)
                     (HostSW ?t ?sw)
                     (Match ?o ?sw ?v))

  :effect (when (ExistVuln ?v ?t)
                (controlling ?t))
)
```

- ***Surveys***

A seguir, são apresentados *surveys* identificados durante a revisão da literatura sobre a utilização do planejamento em IA em teste de software, teste de segurança e teste de intrusão.

Wotawa (2016) apresenta um pequeno *survey* sobre técnicas utilizadas na automatização de teste de segurança. Dentre elas, o autor destaca a utilização do planejamento em IA na modelagem de vulnerabilidades para a realização de testes de intrusão. Este trabalho consta nas publicações identificadas no mapeamento sistemático descrito em A.2.

Grant (2018) apresenta um *survey* sobre planejamento em IA aplicado na automatização de teste de intrusão. Dentre os trabalhos apresentados pelos autores, destacam-se os de Boddy et al. (2005) e Sarraute et al. (2011), já analisados nesta seção. Os autores indicam a obtenção em tempo real de informações das ASTs como uma lacuna identificada na literatura.

Bozic e Wotawa (2019) apresentam, sem uma metodologia de mapeamento sistemático, 19 trabalhos sobre planejamento em IA em teste de software. Os autores classificam estes trabalhos em dois grupos: um sobre teste funcional contendo 12 artigos e outro, com 7 artigos, referente a teste não-funcional (relacionado a requisitos não-funcionais, como desempenho, usabilidade e segurança). Entre os 15 artigos encontrados no mapeamento MSL-Teste, apresentado em A.2, e os 19 artigos identificados por Bozic e Wotawa há uma interseção de 4 artigos: (Mraz et al., 1995), (Howe et al., 1997), (Scheetz et al., 1999) e (von Mayrhauser et al., 2000). Os demais artigos identificados por Bozic e Wotawa e não identificados no mapeamento MSL-Teste são aplicados em áreas como controle de rede, sistemas de rede, protocolos de rede, serviços Web e aplicações Web, industriais e automotivas.

2.6 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou conceitos e definições que serão considerados para a fundamentação teórica da presente pesquisa, além de trabalhos relacionados. Inicialmente, foram apresentados conceitos de teste de software, técnicas de teste e fases de teste. Este trabalho tem como foco a realização de testes durante a fase de teste de sistema, como citado em 2.1, com o objetivo de revelar falhas em aplicações Web, denominadas neste texto como aplicações sob teste (ASTs). Destacou-se também a importância da atividade de planejamento de teste de software para a definição de um plano de teste. O plano de teste de software é um artefato que auxilia a execução dos testes, pois fornece aos testadores informações sobre recursos e ferramental disponíveis.

Em 2.2 foram apresentados conceitos de segurança de software, evidenciando a importância dos testes de segurança para a garantia de qualidade de software. Neste contexto, a técnica

de teste de intrusão foi destacada por ser utilizada neste trabalho. Relacionando os conceitos de teste de software com teste de intrusão, uma vulnerabilidade é a falha que se deseja revelar nas ASTs. As vulnerabilidades mais comuns que podem ser identificadas e exploradas neste tipo de aplicação são descritas no projeto OWASP Top 10, apresentado em 2.2.1. O OWASP Top 10 é atualizado periodicamente, conforme o surgimento de novas vulnerabilidades. Deste modo, observa-se a importância da busca constante por novos métodos de teste de intrusão.

Em 2.2.2 foi destacada a relevância de se empregar uma metodologia na execução de um teste de intrusão. Uma metodologia oferece um sequenciamento de atividades a ser realizado, auxiliando o testador na especificação de testes padronizados. A metodologia PTES foi destacada por ser utilizada como base para os testes realizados neste trabalho. A fim de se obter um maior entendimento sobre o funcionamento das ferramentas de teste de intrusão, foram elaboradas algumas definições acerca destas na Seção 2.3. Estas definições foram realizadas visando a formalização de alguns aspectos das ferramentas, como dados de teste e saída gerada.

O fluxo de teste de intrusão é composto por ferramentas com funcionalidades distintas, conforme apresentado em 2.3. As execuções das ferramentas para obtenção do identificador de sessão e da ferramenta de *crawler* referem-se à instrumentalização do teste. A utilização do identificador de sessão permite que os testes sejam realizados em ASTs com autenticação. Já a execução do *crawler* identifica as URLs da AST utilizadas posteriormente para varredura e exploração. Deste modo, uma URL referente a uma página da AST pode conter dados de teste. As execuções das ferramentas de varredura e exploração referem-se ao teste propriamente dito, ou seja, à identificação e exploração de vulnerabilidades.

Destacou-se também a importância da parametrização nas ferramentas de teste de intrusão. A inclusão de parâmetros permite que os testadores especifiquem aspectos a serem considerados pela ferramenta na execução dos testes, auxiliando na especificação e padronização dos mesmos. Ainda em 2.3, foi descrita a execução sequencial de ferramentas para teste de intrusão. Este uso encadeado de ferramentas identificado na realização de um teste de intrusão indica a possibilidade de representação deste tipo de teste com técnicas de modelagem.

Devido à característica de utilização sequencial de ferramentas, bem como a existência de estado inicial e objetivos do teste bem definidos, percebeu-se que a execução de um teste de intrusão assemelha-se a um problema de planejamento em IA. Logo, a técnica de planejamento em IA, descrita em 2.4, foi considerada para a modelagem apresentada nesta dissertação. Para uma modelagem mais efetiva, percebeu-se a necessidade de inclusão de critérios para a escolha das ferramentas de teste de intrusão mais adequadas para a realização do teste. Para isso, foi selecionada a linguagem PDDL, descrita em 2.4.1, que contém uma funcionalidade de inclusão de variáveis numéricas, que proporcionam a utilização de métricas na geração dos planos. Assim, espera-se que tal funcionalidade desta linguagem seja utilizada como um dos critérios durante a modelagem.

A escolha da linguagem PDDL para este trabalho se deve ainda pela facilidade de modelagem proporcionada pela sua sintaxe. Além disso, trata-se da linguagem padrão em competições atuais de planejamento em IA. Para a geração de planos, optou-se pelo planejador Metric-FF, apresentado em 2.4.3. O Metric-FF foi escolhido por ser um planejador para problemas em PDDL comumente utilizado no estado da arte de planejamento em IA, além de resolver problemas envolvendo variáveis numéricas.

O problema caminho mínimo foi apresentado em 2.4.2 como um exemplo de modelagem de um problema de planejamento em IA, culminando na descrição do problema e domínio em arquivos PDDL. Para a obtenção do plano, tais arquivos foram utilizados como entrada do planejador Metric-FF. A elaboração deste exemplo permitiu a familiarização com a sintaxe e funcionalidades da linguagem PDDL, como operações com variáveis numéricas. Além disso,

foi possível observar o funcionamento do Metric-FF. Ao término desta modelagem, foi obtida a descrição do problema caminho mínimo como a quádrupla (P, I, O, A), conforme a definição de planejamento em IA apresentada em 2.4.

Uma revisão da literatura foi realizada com o intuito de verificar as abordagens de utilização do planejamento em IA na engenharia de software e em teste de intrusão. Inicialmente, foram realizados dois mapeamentos sistemáticos da literatura (MSLs), apresentados no Apêndice A. Optou-se pela metodologia de Petersen et al. (2015), pois estes autores realizam um MSL na área de engenharia de software. O mapeamento sistemático se mostrou uma forma eficaz para identificar, avaliar e interpretar pesquisas disponíveis relevantes para uma questão de pesquisa particular. Os mapeamentos foram conduzidos com viés quantitativo, buscando identificar lacunas de pesquisa nestas áreas de estudo.

O mapeamento MSL-ReqDes, apresentado em A.1, investigou a utilização do planejamento em IA em engenharia de requisitos e design. Já o MSL-Teste, descrito em A.2, buscou publicações sobre planejamento em IA aplicado a teste de software. Com a realização dos mapeamentos foi possível responder as questões de pesquisa principais especificadas em cada protocolo. Respondendo a pergunta de pesquisa do MSL-ReqDes, "existem trabalhos onde planejamento em IA é utilizado nas fases engenharia de requisitos e design?", foram identificadas publicações que aplicam o planejamento em IA em ambas as fases de desenvolvimento com propostas de técnica, métodos e *frameworks*. Respondendo a pergunta do MSL-Teste, "como o planejamento em IA é utilizado em teste de software?", foram identificadas propostas de modelos, técnicas, *frameworks* e ferramentas.

A extração de dados realizada nas publicações selecionadas nos mapeamentos respondeu as demais subquestões de pesquisa especificadas em cada protocolo. Como uma síntese dos resultados obtidos, foram geradas duas tabelas: a Tabela A.5 que resume as informações do MSL-ReqDes e a Tabela A.10 que contém informações do MSL-Teste. Algumas informações subentendidas nas publicações foram adicionadas nestas tabelas, como por exemplo a fase de teste na Tabela A.10. Outras informações, como técnica de planejamento em IA, não estavam presentes em algumas publicações e foram adicionadas nas duas tabelas após pesquisa adicional.

Com a análise das Tabelas A.5 e A.10 identificou-se lacunas nos dados extraídos, evidenciando algumas deficiências nas propostas encontradas na literatura. Com a análise dos dados do MSL-ReqDes, foi possível observar a baixa utilização do planejamento em IA em engenharia de requisitos nas publicações selecionadas. Além disso, não foram constatadas propostas de ferramentas para as fases engenharia de requisitos e design.

Por meio da análise dos resultados obtidos com o mapeamento MSL-Teste, observou-se que nenhuma das publicações especificam explicitamente a fase de teste onde as propostas são realizadas. Além disso, constatou-se nas publicações selecionadas o uso mais frequente do planejamento em IA no teste funcional em relação às técnicas de teste estrutural e baseada em erros. Foi possível observar também a recente aplicação do planejamento em IA em teste de intrusão. Isto motivou a procura de outras publicações referentes a este tópico.

Dentre as publicações identificadas no mapeamento MSL-Teste, duas utilizam planejamento em IA em teste de intrusão. Em geral, os autores utilizam termos como "modelagem de ataque", "grafo de ataque" e "caminho de ataque" para se referir ao uso do planejamento em IA neste tipo de teste. Além disso, referem-se comumente a teste de intrusão como "*pentesting*". Tais termos fogem do escopo do mapeamento, justificando o número de publicações identificadas. Os trabalhos relacionados da Seção 2.5 foram identificados em busca manual com estes termos.

Foram identificadas duas abordagens de utilização de planejamento em IA em teste de intrusão nos trabalhos relacionados descritos em 2.5. A primeira abordagem identificada refere-se à modelagem em PDDL de vulnerabilidades como um problema de planejamento.

Nestas publicações, o plano gerado representa a sequência de passos que atestam a existência da vulnerabilidade na aplicação sob teste (AST). A outra abordagem identificada diz respeito à modelagem em PDDL de *exploits* como ações do problema de planejamento. Desta forma, os planos gerados nestas publicações representam sequências de *exploits* necessários para exploração da AST.

Portanto, esta dissertação de mestrado difere-se das abordagens identificadas na literatura por realizar a modelagem em PDDL do problema de planejamento em IA representando o fluxo de execução dos diferentes tipos de ferramentas utilizados em teste de intrusão. Além disso, características da AST também são consideradas para a modelagem. Desta maneira, o plano gerado representa a sequência de execuções de ferramentas necessária para a instrumentalização do teste, a varredura e a exploração de vulnerabilidades na AST.

Associando os conceitos de teste de software e planejamento em IA, a definição e modelagem do problema de planejamento em IA proposto nesta dissertação referem-se à atividade de planejamento de teste de software, como citado em 2.1. Já o plano obtido com o planejador associa-se ao plano de teste de software, ao indicar ao testador as configurações das ferramentas necessárias para a realização de um teste de intrusão para um determinado cenário de teste.

As ferramentas utilizadas para uso no método desta dissertação foram definidas a partir de um estudo exploratório, apresentado no capítulo seguinte.

3 ESTUDO EXPLORATÓRIO COM FERRAMENTAS DE TESTE DE INTRUSÃO

Este capítulo apresenta um estudo exploratório com ferramentas de teste de intrusão. A Seção 3.1 apresenta a metodologia elaborada para a realização do estudo. A Seção 3.2 apresenta uma análise das ferramentas de teste de intrusão selecionadas. Em seguida, a Seção 3.3 apresenta uma análise das aplicações sob teste (ASTs) selecionadas para o estudo. Por fim, a Seção 3.4 apresenta considerações do capítulo.

3.1 METODOLOGIA

Este estudo tem como objetivo definir as ferramentas de teste de intrusão que serão utilizadas no método proposto nesta dissertação. O estudo consiste em analisar funcionalidades e usabilidade de ferramentas de teste de intrusão, bem como investigar o comportamento das ferramentas em execuções com diferentes tipos de ASTs. O estudo exploratório é composto de quatro atividades distintas: (i) análise das ferramentas de teste de intrusão, (ii) análise das ASTs, (iii) instalação das ferramentas e (iv) execução das ferramentas. Todas as atividades são realizadas pelo testador.

A Figura 3.1 apresenta a sequência de atividades da metodologia. A descrição de cada atividade é realizada a seguir.

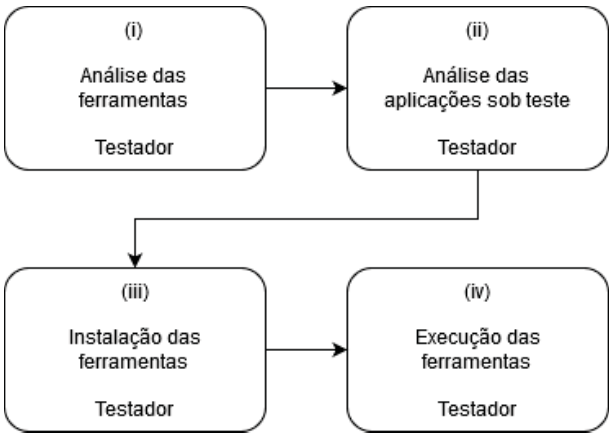


Figura 3.1: Metodologia do estudo exploratório. Fonte: O autor (2019).

- Atividade (i): nesta atividade é realizada a análise das ferramentas de teste de intrusão selecionadas. O objetivo desta atividade é obter uma visão geral das características e funcionalidades de cada ferramenta. Primeiramente, são obtidas características das ferramentas, como linguagem de implementação e sistema operacional. Em seguida, são identificadas as principais funcionalidades de cada ferramenta, como forma de execução e vulnerabilidades identificadas em suas execuções. Todas as informações obtidas são catalogadas e analisadas.

- Atividade (ii): nesta atividade é realizada a análise das ASTs selecionadas para este estudo. Inicialmente, cada AST é configurada em ambiente *docker*[1]. A análise desta

[1]*Docker* é uma ferramenta para empacotamento de aplicações em ambiente virtual. Disponível em: https://www.docker.com/

atividade consiste na identificação das vulnerabilidades existentes nas ASTs, objetivando a realização posterior de testes de *benchmark* com as ferramentas selecionadas.

- Atividade (iii): nesta atividade é realizada a instalação das ferramentas de teste de intrusão selecionadas e analisadas na atividade (i). Esta atividade é realizada em uma máquina virtual configurada nos servidores do Departamento de Informática (DInf) da UFPR. Isto justifica-se pelo fato de proporcionar padronização do ambiente de execução dos testes e, consequentemente, nos resultados obtidos.

- Atividade (iv): nesta atividade são realizadas execuções das ferramentas de teste de intrusão selecionadas. Estes testes são realizados nos computadores do DInf, com sistema operacional Linux Mint 19.1 Cinnamon, 4.00GB de memória 4.0.10m e processador Intel Core i5-2400 CPU @ 3.10GHz × 4. As execuções são realizadas nas ASTs analisadas na atividade (ii).

  O objetivo principal desta atividade é identificar a possibilidade de utilização encadeada das ferramentas de teste de intrusão selecionadas, conforme fluxo descrito em 2.3. Pretende-se também analisar a usabilidade das ferramentas quando executadas com diferentes tipos de AST. As especificidades de cada AST selecionada possibilitam que configurações distintas das ferramentas sejam exercitadas. Os parâmetros necessários para a execução de cada configuração das ferramentas são identificados e catalogados. A definição das ferramentas para uso no método proposto é realizada ao término desta atividade.

3.2 ANÁLISE DAS FERRAMENTAS

A amostra de ferramentas de teste de intrusão utilizada neste estudo exploratório é oriunda da dissertação de mestrado "Web (Eternamente) Revisitada: Análise de Vulnerabilidades Web e de Ferramentas de Código Aberto para Exploração" (Neto, 2019), do laboratório Fundamentos e Aplicações em Engenharia de Software (Lab FAES) do Programa de Pós-Graduação em Informática (PPGInf) da UFPR. Considera-se a utilização desta amostra pré-definida como uma forma de continuidade dos estudos realizados neste laboratório de pesquisa.

O critério de seleção de ferramentas foi definido após análise das seis edições do projeto OWASP Top 10, disponibilizadas entre os anos 2003 e 2017. Com esta análise, constatou-se que vulnerabilidades sobre validação de dados de entrada, como injeção de SQL e *cross-site scripting* (XSS), são as mais frequentes em aplicações Web. Desta forma, a seleção restringiu-se a ferramentas que detectam e/ou exploram estas vulnerabilidades.

A amostra de ferramentas selecionadas é composta por Arachni (2019), *Browser Exploitation Framework* (BeEF, 2019), HTCAP (2019), *Iron Web Application Advanced Security Testing Platform* (IronWASP, 2019), Skipfish (2019), SQLmap (2019), Metasploit (2019), Vega (2019), Wapiti (2019), *Web Application Attack and Audit Framework* (w3af, 2019), Wfuzz (2019), *XSS Exploit Framework* (Xenotix, 2019), *Cross Site Scripter* (XSSer, 2019) e *Zed Attack Proxy* (ZAP, 2019). Todas as ferramentas da amostra são atualizadas e mantidas por comunidades que prestam manutenção periódica.

As ferramentas que compõem a amostra são de código aberto. Isto proporciona acesso ao código-fonte das ferramentas, permitindo análise mais aprofundada de certas funcionalidades, quando necessário. Existem ferramentas comerciais, como *Acunetix*[2] e *Burp Suit*[3], com

[2]Disponível em: https://www.acunetix.com
[3]Disponível em: https://portswigger.net

funcionalidades que se assemelham às ferramentas selecionadas para o estudo. Entretanto, tais ferramentas são desconsideradas por não permitirem acesso a todas as funcionalidades em suas versões gratuitas.

Como descrito na atividade (i) da metodologia, são obtidas informações das ferramentas selecionadas por meio da análise das documentações disponibilizadas por seus desenvolvedores. A Tabela 3.1 sintetiza as informações obtidas com a realização desta atividade. Respectivamente nas colunas "Ferramenta", "Linguagem" e "SO" são indicados nome da ferramenta, linguagem em que a ferramenta foi implementada e sistema operacional de execução da ferramenta. A coluna "Tipo" classifica a ferramenta quanto à varredura, exploração e *framework*. Na coluna "Execução" é indicado se a ferramenta é executada por linha de comando (LC) ou em interface gráfica (IG). Na coluna "Vulnerabilidade" são identificadas as vulnerabilidades presentes no OWASP Top 10 2017 que a ferramenta identifica e/ou explora. Por fim, a coluna "Saída" contém os formatos de arquivo de saída disponibilizados pelas ferramentas. As marcações "-" referem-se a informações que não se aplicam às ferramentas.

Tabela 3.1: Síntese de informações das ferramentas de teste de intrusão. Fonte: O autor (2019).

Ferramenta	**Linguagem**	**SO**	**Tipo**	**Execução**	**Vulnera-bilidade**	**Saída**
Arachni	Ruby	Linux	Varredura	LC	Injeção de SQL e XSS	HTML, XML, JSON e txt
BeEF	Ruby	Linux	*Framework*	IG	XSS	txt
HTCAP	Python	Linux	Varredura	LC	Injeção de SQL, XSS e XXE	JSON e db
IronWASP	Phyton e Ruby	Windows	Varredura	IG	Injeção de SQL e XSS	HTML e RTF
Skipfish	C	Linux	Varredura	LC	Injeção de SQL, XSS e XXE	HTML
SQLmap	Python	Linux	Varredura e exploração	LC	Injeção de SQL	CSV, HTML e SQLite
Metasploit	Ruby	Linux	*Framework*	LC	-	txt
Vega	Java	Linux	Varredura	IG	Injeção de SQL e XSS	txt
Wapiti	Phyton	Linux	Varredura	LC	Injeção de SQL, XSS e XXE	JSON
W3af	Python	Linux	*Framework*	IG	Injeção de SQL e XSS	XML e txt
Wfuzz	Phyton	Linux	*Framework*	LC	-	txt
Xenotix	.NET	Windows	Varredura e exploração	IG	XSS	txt

Continua na próxima página

Tabela 3.1 - *Continua da página anterior*

Ferramenta	Linguagem	SO	Tipo	Execução	Vulnera-bilidade	Saída
XSSer	Python	Linux	Varredura e exploração	IG e LC	XSS	XML
ZAP	Java	Linux	Varredura	IG e LC	Injeção de SQL, XSS	TML, XML e JSON

A seguir, são realizadas análises das informações contidas na Tabela 3.1.

- **Linguagem e sistema operacional**

Em relação às linguagens de implementação, nota-se o predomínio das linguagens Python, utilizada em 50% das ferramentas da amostra, e Ruby, presente em 28% das ferramentas. As linguagens C, Java e .NET também são utilizadas, embora com menor frequência. As ferramentas da amostra, em sua maioria (85%), são implementadas para o sistema operacional Linux. O restante da amostra é executada no sistema operacional Windows.

- **Tipo de ferramenta**

Quanto ao tipo de ferramenta, observa-se na amostra as funcionalidades de varredura, exploração e *frameworks* para teste de intrusão. Entre as ferramentas selecionadas, os tipos mais frequentes são as de varredura, que representam 50% do total. Também constata-se na amostra ferramentas com funcionalidades de varredura e exploração para duas vulnerabilidades específicas: SQLmap, que explora a vulnerabilidade injeção de SQL, e as ferramentas Xenotix e XSSer, para exploração de XSS. Na amostra constam também os *frameworks* BeEF, Metasploit, W3af e Wfuzz.

- **Forma de execução**

No que diz respeito à execução das ferramentas da amostra, constata-se execuções por meio de linha de comando e em interface gráfica. As execuções realizadas por linha de comando correspondem a 50% da amostra. Já as ferramentas executadas unicamente em interface gráfica equivalem a 35% da amostra. As ferramentas XSSer e ZAP realizam execuções em interface gráfica, porém também oferecem opções de execução por linha de comando. A análise da forma de execução das ferramentas resultou em um relatório[4] contendo a catalogação dos parâmetros disponibilizados pelas ferramentas executadas por linha de comando. Este relatório é utilizado como auxílio na posterior realização dos testes.

- **Vulnerabilidades**

Como esperado devido ao critério de seleção, percebe-se o predomínio das vulnerabilidades injeção de SQL e XSS nas ferramentas da amostra. Destaca-se também a presença da vulnerabilidade XXE, contida no OWASP Top 10 2017. Todas as ferramentas de varredura da amostra são capazes de identificar ao menos duas destas vulnerabilidades. Além das vulnerabilidades citadas, todas as ferramentas da amostra realizam testes em outras vulnerabilidades que

[4]Disponível em: https://github.com/lf-lima/RelatorioTesteIntrusao

não estão presentes no Owasp Top 10 2017. No entanto, tais vulnerabilidades não constam na catalogação realizada por fugir do escopo do estudo.

As ferramentas de exploração são implementadas para testes em variações de uma mesma vulnerabilidade. Por exemplo, a ferramenta SQLmap explora diferentes tipos da vulnerabilidade injeção de SQL, como *blind SQL injection*, *boolean-based*, *time-based* e *error-based*. Já as ferramentas Xenotix e XSSer exploram variações da vulnerabilidade XSS, como *reflected* XSS e *DOM based* XSS.

Os *frameworks* contidos na amostra que contém funcionalidades que se diferem das demais ferramentas selecionadas são indicados com a marcação "-" na coluna "Vulnerabilidades" da Tabela 3.1. O *framework* Wfuzz atua como um *fuzzer*, ou seja, realiza testes nos parâmetros da AST em busca de *buffer overflows*, por exemplo. Já o *framework* Metasploit possui um conjunto de *exploits* que, ao serem executados na AST, podem explorar uma vulnerabilidade existente.

- **Formato de saída**

Geralmente, as ferramentas para teste de intrusão disponibilizam formatos variados para o relatório gerado após a realização do teste. Os formatos dos arquivos de saída são selecionados com a utilização de parâmetros específicos de cada ferramenta. Nas ferramentas selecionadas, observa-se o predomínio do formato txt, disponibilizado por 50% da amostra. Considera-se como formato txt os casos em que o conteúdo do relatório gerado pela execução do teste é exibido diretamente no terminal ou na interface gráfica da ferramenta. Ou seja, quando não ocorre o redirecionamento da saída para um arquivo específico. Os formatos HTML, XML e JSON também são frequentes na amostra, constando em 28% das ferramentas. Os formatos db, TML, CSV, SQLite e RTF aparecem como opções nas ferramentas, embora com frequência reduzida.

3.3 ANÁLISE DAS APLICAÇÕES SOB TESTE

As aplicações sob teste (ASTs) selecionadas para este estudo exploratório são aplicações Web vulneráveis, isto é, implementadas com vulnerabilidades pré-estabelecidas. Ou seja, contém um conjunto V_{ast} de vulnerabilidades, conforme definição realizada em 2.3. Desta forma, estas ASTs selecionadas tratam-se de aplicações para *benchmark*, pois é possível realizar a comparação das vulnerabilidades existentes na aplicação com as que foram identificadas com a execução de uma ferramenta de teste de intrusão. As aplicações selecionadas são configuráveis em ambiente *docker*, permitindo que os testes aconteçam em um ambiente controlado e configurável pelo testador. Estas características das ASTs selecionadas auxiliam os testadores na análise dos resultados obtidos com a execução de testes.

A amostra de ASTs contém sete aplicações selecionadas objetivando a realização de execuções de testes com diferentes configurações das ferramentas. A partir dos critérios de seleção mencionados, as ASTs selecionadas são Badstore[5], Webgoat[6], *Damn Vulnerable Web Application* (DVWA)[7], *buggy Web Application* (bWAPP)[8], Mutillidae[9], *Xtreme Vulnerable Web Application* (XVWA)[10] e BTSLab[11]. Badstore, BTSLab, Mutillidae e XVWA são ASTs que não requerem autenticação prévia. Já bWAPP, DVWA e Webgoat são ASTs que necessitam de

[5]Configurada em: http://testesseg.c3sl.ufpr.br:3000
[6]Configurada em: http://testesseg.c3sl.ufpr.br:3001
[7]Configurada em: http://testesseg.c3sl.ufpr.br:3002
[8]Configurada em: http://testesseg.c3sl.ufpr.br:3003
[9]Configurada em: http://testesseg.c3sl.ufpr.br:3004
[10]Configurada em: http://testesseg.c3sl.ufpr.br:3005
[11]Configurada em: http://testesseg.c3sl.ufpr.br:3006

um processo de autenticação para sua utilização. Estas especificidades das ASTs selecionadas permitem que as execuções ocorram com diferentes configurações das ferramentas. Assim, é possível observar o comportamento das ferramentas em diferentes cenários de teste.

Como parte da atividade (ii) da metodologia, as vulnerabilidades contidas em cada AST selecionada foram identificadas por meio da análise da documentação e demais informações disponibilizadas pelos desenvolvedores das aplicações. Esta análise resultou na síntese apresentada na Tabela 3.2. As colunas "V1" a "V10", referem-se às vulnerabilidades do OWASP Top 10 2017, conforme apresentadas em 2.2.1. A coluna "Outras" refere-se às vulnerabilidades que não fazem parte da edição do OWASP Top 10 mencionada. A existência de uma vulnerabilidade na AST é indicada com o marcador "✓".

Tabela 3.2: Vulnerabilidades contidas nas aplicações sob teste. Fonte: O autor (2019).

AST	**Vulnerabilidade**										
	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	Outras
Badstore	✓						✓				✓
BTSLab	✓						✓				✓
bWAPP	✓	✓	✓	✓	✓	✓	✓		✓		✓
DVWA	✓						✓				✓
Mutillidae	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
XVWA	✓						✓				✓
Webgoat	✓	✓		✓	✓		✓	✓	✓		✓

Analisando a Tabela 3.2 nota-se que todas as ASTs selecionadas foram implementadas com as vulnerabilidades injeção de SQL (V1) e XSS (V7). As demais vulnerabilidades do OWASP Top 10 2017 constam nas ASTs, ainda que com menor frequência. Dentre estas, destaca-se a presença das vulnerabilidades "quebra de autenticação", "XXE", "quebra de controle de acesso" e "utilização de componentes vulneráveis" (V2, V3, V5 e V7, respectivamente).

Após a configuração das ASTs, foi realizada a instalação das ferramentas de teste de intrusão descritas na seção anterior, conforme estabelecido na atividade (iii) da metodologia. Todas as ferramentas de teste de intrusão da amostra foram executadas tendo como entrada ao menos uma AST selecionada. A catalogação das vulnerabilidades presentes nas ASTs proporcionou a realização de testes de *benchmark* com as ferramentas selecionadas, sendo este um dos intuitos da atividade (iv) da metodologia.

3.4 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou um estudo exploratório com ferramentas de teste de intrusão. Para a realização do estudo foram selecionadas 14 ferramentas e *frameworks* de teste de intrusão e 7 aplicações sob teste (ASTs). As ferramentas foram escolhidas para o estudo por realizarem testes das vulnerabilidades mais frequentes no OWASP Top 10. Além disso, todas as ferramentas são de código aberto e submetidas à manutenção periódica por seus desenvolvedores. Já as ASTs foram selecionadas por serem vulneráveis, possibilitando testes de *benchmark*. Além disso, as ASTs selecionadas são configuráveis em ambiente *docker*, permitindo flexibilidade de onde os testes podem ser realizados e, desta maneira, contribuindo para a padronização dos mesmos.

A análise apresentada em 3.2 buscou identificar as principais características e funcionalidades das ferramentas e *frameworks* de teste de intrusão da amostra selecionada. Com esta análise, constatou-se o predomínio da linguagem Python na implementação das ferramentas da amostra. Observou-se que estas ferramentas são executadas, em sua maioria, por meio de linha

de comando no sistema operacional Linux. Nas ferramentas da amostra foi possível identificar as funcionalidades de varredura e exploração, sendo injeção de SQL e XSS as vulnerabilidades identificadas e/ou exploradas pela maioria das ferramentas. Percebeu-se ainda que o formato txt é o mais empregado pelas ferramentas da amostra na geração de seus relatórios de execução.

A análise da amostra de ASTs, apresentada em 3.3, consistiu na identificação das vulnerabilidades da OWASP Top 10 2017 presentes em cada aplicação. Com esta análise foi possível identificar a presença constante das vulnerabilidades injeção de SQL e XSS nas ASTs selecionadas. A catalogação de vulnerabilidades presentes nas ASTs proporcionou a realização posterior de testes de *benchmark* com as ferramentas selecionadas. Após a instalação das ferramentas e configuração das ASTs, foram realizadas execuções de testes para se observar o comportamento das ferramentas. Durante a realização das execuções das ferramentas foram observados aspectos como processo de autenticação e encadeamento das ferramentas.

A amostra de ASTs contém aplicações que não requerem autenticação e outras que necessitam de uma autenticação prévia para sua utilização. Desta forma, foi possível constatar a necessidade de utilização de configurações distintas das ferramentas para as execuções dos testes, de acordo com as especificidades de cada tipo de AST. As execuções das diferentes configurações das ferramentas de teste de intrusão diferem-se por meio da utilização de parâmetros distintos. A principal característica que distingue as configurações das ferramentas é a existência do processo de autenticação na AST. Para a execução dos testes em AST com autenticação, além da URL da aplicação, são necessários parâmetros para indicar usuário e senha ou identificador de sessão. Por exemplo, a ferramenta HTCAP utiliza o parâmetro "`-c`" para informar o identificador de sessão da AST. Em AST sem autenticação este parâmetro não é utilizado, caracterizando uma configuração distinta da ferramenta.

Como parte dos objetivos do estudo, buscou-se identificar a possibilidade do uso encadeado de ferramentas de teste de intrusão da amostra. Para isso, a documentação das ferramentas foi analisada, a fim de se investigar todas as funcionalidades disponibilizadas por cada ferramenta. Esta análise teve como intuito selecionar na amostra ferramentas para cada etapa do fluxo de execução de teste de intrusão, descrito em 2.3. Logo, buscou-se ferramentas com funcionalidades de obtenção de identificador de sessão, *crawler*, varredura de aplicação e exploração de vulnerabilidades.

Todas as ferramentas da amostra foram executadas em ao menos uma das ASTs selecionadas. A definição das ferramentas para uso no método desta dissertação foi realizada de acordo com aspectos observados nas ferramentas durante estas execuções. Foram selecionadas para o método ferramentas com funcionalidades que atendam ao fluxo do teste de intrusão, priorizando ferramentas executadas por linha de comando e no sistema operacional Linux. Estes critérios de seleção foram definidos com o objetivo de realizar a posterior integração das ferramentas e automatização da execução dos testes. Desta forma, ferramentas para o sistema operacional Windows e com execuções apenas em interface gráfica foram desconsideradas para a modelagem com planejamento em IA. A Tabela 3.3 apresenta os critérios empregados para a inclusão ou exclusão de cada ferramenta.

Tabela 3.3: Critérios para definição das ferramentas de teste de intrusão para uso no método. Fonte: O autor (2019).

Ferramenta	**Critério para inclusão**	**Critério para exclusão**
Arachni	Varredura de aplicação	-
BeEF	-	*Framework*
HTCAP	*Crawler*	-

Continua na próxima página

Tabela 3.3 - *Continua da página anterior*

Ferramenta	**Critério para inclusão**	**Critério para exclusão**
IronWASP	-	Windows
Skipfish	Varredura de aplicação	-
SQLmap	Exploração de injeção de SQL	-
Metasploit	*Framework* de *exploits*	-
Vega	-	Interface gráfica
Wapiti	Obtenção do identificador de sessão	-
W3af	-	*Framework*
Wfuzz	-	*Framework*
Xenotix	-	Windows
XSSer	Exploração de XSS	-
ZAP	Varredura de aplicação	-

Para a obtenção do identificador de sessão foi selecionada a Wapiti-getcookie, extensão da ferramenta Wapiti com esta finalidade. A ferramenta HTCAP foi selecionada como *crawler*. Para varredura de aplicação foram selecionadas as ferramentas Arachni, Skipfish e ZAP. Quanto à exploração, foram selecionadas as ferramentas SQLmap, para exploração de injeção de SQL, e XSSer, para exploração de XSS. As ferramentas IronWASP e Xenotix não foram consideradas por serem executadas no sistema operacional Windows. Já a ferramenta Vega foi desconsiderada por ser executada apenas em interface gráfica. Além dos critérios já mencionados, estas ferramentas foram selecionadas para uso no método proposto pois atendem à metodologia PTES, utilizada para a execução dos testes nesta dissertação.

O *framework* Metasploit foi selecionado para uso no método proposto pois contém a funcionalidade de inclusão de novos *exploits* em sua base local. Esta funcionalidade do Metasploit foi considerada para os casos em que não há opção de ferramenta de exploração disponível para uma determinada vulnerabilidade identificada. Os *frameworks* BeEF, W3af e Wfuzz foram desconsiderados pois suas funcionalidades fogem do escopo esperado para a modelagem e execução de teste de intrusão desta dissertação.

O capítulo seguinte apresenta o método de teste de intrusão proposto.

4 MÉTODO DE TESTE DE INTRUSÃO COM PLANEJAMENTO EM IA

Este capítulo apresenta um método de teste de intrusão composto de atividades de planejamento do teste e execução do teste. Inicialmente, a Seção 4.1 apresenta uma visão geral do método proposto. A Seção 4.2 apresenta a atividade de planejamento do teste. A Subseção 4.2.1 apresenta a etapa de definição do problema de planejamento em IA. Em seguida, a Subseção 4.2.2 apresenta a etapa para modelagem em PDDL dos arquivos de domínio e problema. Finalizando a Seção 4.2, a Subseção 4.2.3 apresenta a etapa de geração dos planos de teste para cada cenário de teste estabelecido. A Seção 4.3 apresenta a atividade de execução do teste. A Subseção 4.3.1 descreve a etapa para instrumentalização do teste. As Subseções 4.3.2 e 4.3.3 apresentam as etapas para execução das ferramentas de varredura de aplicação e exploração, respectivamente. Em seguida, a Subseção 4.3.4 descreve uma proposta de módulo automatizável para busca de *exploits* e atualização do *framework* utilizado no método. Por fim, a Seção 4.4 apresenta considerações do capítulo.

4.1 VISÃO GERAL

Utilizando as noções de teste de software, como citado em 2.1, o método de teste de intrusão proposto é dividido em atividades de planejamento e de execução do teste. O método inicia-se com a atividade de planejamento do teste que objetiva a geração de planos de teste para um determinado cenário. Esta atividade é composta de três etapas realizadas pelo testador: 1) o testador estabelece os estados que compõem o problema de planejamento em IA e que representam o fluxo de execução das ferramentas de teste de intrusão; 2) o testador realiza a modelagem em PDDL do problema definido na etapa 1, gerando manualmente arquivos de domínio e de problema conforme requerido por esta linguagem; e 3) geração dos planos de teste, na qual o testador encaminha os arquivos modelados na etapa 2 a um planejador que gera os planos automaticamente.

A atividade de execução do teste ocorre após a conclusão da atividade de planejamento do teste. A atividade de execução do teste objetiva a associação das informações contidas nos planos de teste obtidos no planejamento do teste com módulos de código automatizados para a execução de testes de intrusão. Esta atividade é composta de quatro etapas que contém módulos para integração das ferramentas de teste de intrusão e automatização da execução das mesmas: 1) execução das ferramentas de instrumentalização do teste, que objetiva a obtenção de informações da aplicação sob teste (AST), como identificador de sessão e páginas, que são utilizadas nas etapas posteriores; 2) execução da ferramenta de varredura de aplicação, realizada com o objetivo de detectar vulnerabilidades na AST; 3) execução das ferramentas de exploração com o objetivo de obtenção de acesso à AST a partir das vulnerabilidades detectadas na etapa anterior; e 4) geração automática de um relatório final do teste realizado. Para a execução destas etapas são propostos neste trabalho *scripts* adaptáveis de acordo com o plano de teste gerado para o cenário de teste estabelecido.

Ainda como parte da atividade de execução do teste, existe uma etapa intermediária para a busca de *exploits*. A realização desta etapa ocorre em casos em que não há uma ferramenta de exploração específica para determinada vulnerabilidade detectada na etapa 2, de varredura. Apresenta-se, para esta etapa intermediária, uma proposta de módulo automatizável para a realização da busca de *exploits* em uma base de dados, objetivando a exploração da vulnerabilidade identificada. Neste módulo proposto também é realizada a atualização de uma

base de dados local do *framework* de teste de intrusão, o qual é incluído no método como alternativa às ferramentas de exploração. Nestes casos, a execução do *framework* resulta no relatório final do teste.

As atividades de planejamento do teste e de execução do teste, bem como a descrição de suas etapas, são detalhadas em 4.2 e 4.3, respectivamente.

4.2 PLANEJAMENTO DO TESTE

O método de teste de intrusão proposto inicia-se com a atividade de planejamento do teste que contém as etapas a seguir.

- **Definição do problema de planejamento em IA**: nesta etapa, o testador estabelece os estados que representam o fluxo de execução de um teste de intrusão. Esta etapa é apresentada em 4.2.1;
- **Modelagem do problema de planejamento em IA**: nesta etapa, o testador realiza a modelagem do problema definido na etapa anterior em arquivos de problema e domínio em PDDL. Esta etapa é apresentada em 4.2.2;
- **Geração dos planos de teste**: nesta etapa, o testador utiliza os arquivos em PDDL modelados na etapa anterior como entrada de um planejador que gera os planos de teste para cada cenário de teste definido. Em 4.2.3 é apresentada a descrição desta etapa.

4.2.1 Definição do problema de planejamento

A atividade de planejamento do teste inicia-se com a etapa definição do problema de planejamento em IA. No método proposto, o problema de planejamento representa a execução das ferramentas utilizadas em um teste de intrusão. O fluxo de execução das ferramentas de teste de intrusão, descrito na Seção 2.3, foi utilizado como base para a definição do problema. Define-se, para este problema, quatro estados distintos caracterizados pelas execuções de ferramentas para *crawler*, varredura e exploração, que compõem o fluxo principal de execução de um teste de intrusão.

Ilustrando o problema, temos o diagrama de estados apresentado na Figura 4.2. O estado inicial $\mathbf{E_i}$, os estados intermediários $\mathbf{Int_1}$ e $\mathbf{Int_2}$, o estado final $\mathbf{E_f}$ e os arcos do diagrama são descritos a seguir.

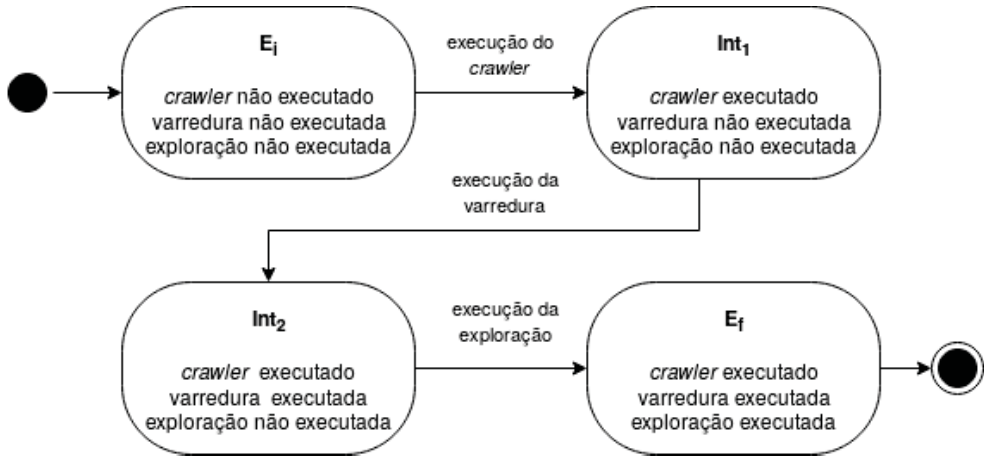


Figura 4.1: Estados do problema de planejamento em IA para teste de intrusão. Fonte: O autor (2019).

- **E_i (estado inicial)**: neste estado, todas as ferramentas que compõem um teste de intrusão (*crawler*, varredura de aplicação e exploração de vulnerabilidades) ainda devem ser executadas na aplicação sob teste (AST). Logo, este estado inicial representa o momento anterior ao início do teste.

- **Int_1 (estado intermediário 1)**: a execução da ferramenta para *crawler* inicia o teste de intrusão. Esta execução leva o problema para o primeiro estado intermediário, representado no diagrama por **Int_1**. Ou seja, no estado **Int_1**, de acordo com o fluxo de teste de intrusão, as ferramentas de varredura e exploração ainda devem ser executadas.

- **Int_2 (estado intermediário 2)**: o passo seguinte do teste de intrusão é a execução da varredura nas páginas da AST identificadas pelo *crawler*. A execução da ferramenta de varredura representa a transição do estado **Int_1** para o segundo estado intermediário **Int_2**. Logo, no estado **Int_2**, apenas a ferramenta de exploração ainda deve ser executada.

- **E_f (estado final)**: o teste de intrusão se encerra com a execução de uma ferramenta de exploração em páginas da AST com vulnerabilidades identificadas na varredura. A execução da ferramenta de exploração indica a transição do estado intermediário **Int_2** para o estado final **E_f**. Desta forma, o estado **E_f** é caracterizado pelas execuções das ferramentas para *crawler*, varredura e exploração já realizadas. Logo, este estado final equivale a dizer que o teste de intrusão está finalizado.

- **Arcos**: os arcos do diagrama representam a transição entre os estados definidos. Os rótulos de cada arco indicam o tipo de ação que deve ser executada para ocorrer a transição. Desta forma, as ações contidas na modelagem do problema em PDDL, apresentada na seção seguinte, representam as execuções de ferramentas de *crawler*, varredura e exploração. Assim, de acordo com o problema definido, um possível plano para a resolução do problema consiste na execução encadeada de ações destes três tipos.

A Figura 4.2 relaciona o fluxo de execução das ferramentas de teste de intrusão, conforme apresentado em 2.3, com os estados definidos para o problema de planejamento.

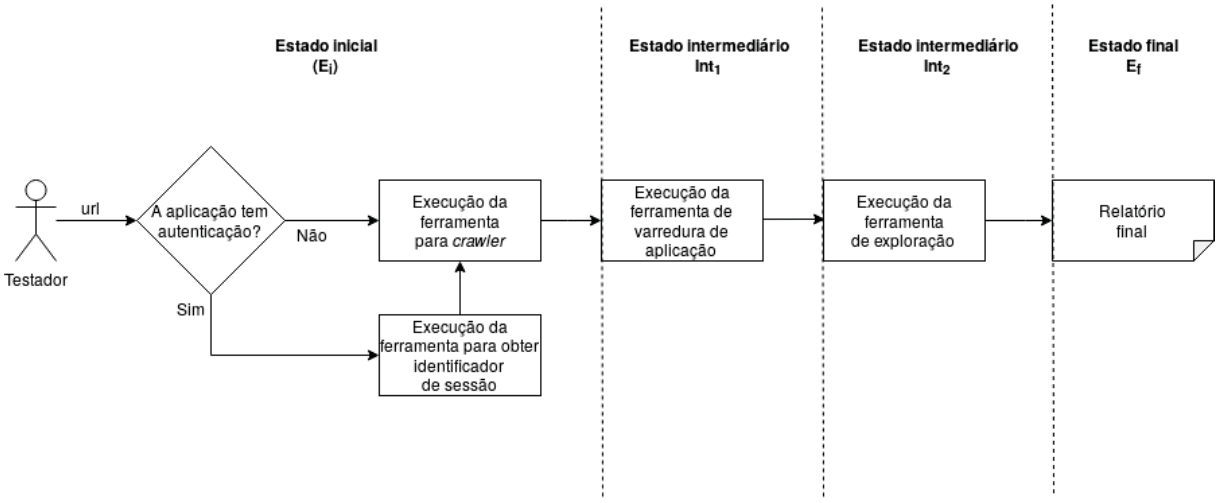


Figura 4.2: Estados associados ao fluxo das ferramentas de teste de intrusão. Fonte: O autor (2019).

Conforme descrito anteriormente, o estado inicial **E_i** caracteriza o início do teste, ou seja, ao momento anterior às execuções das ferramentas para *crawler*, varredura e exploração. A execução do *crawler* indica a transição para o estado intermediário **Int_1**. Além da execução

do *crawler*, neste estado ocorre a execução de uma ferramenta específica para obtenção de identificador de sessão. Ou seja, no estado $\mathbf{E_i}$ são realizadas execuções de ferramentas de instrumentalização do teste.

A ferramenta para obtenção do identificador de sessão é executada somente em casos onde a AST requer autenticação prévia para sua utilização. A execução desta ferramenta também será modelada como uma ação do problema de planejamento. Assim, nestes casos, o plano gerado contém esta ação adicional, além das outras que representam a execução das demais ferramentas que compõem o fluxo principal de um teste de intrusão.

No estado $\mathbf{Int_1}$, a ferramenta para *crawler* já foi executada, bem como a ferramenta para obter o identificador de sessão, quando necessária. Então, neste estado ocorre a execução da ferramenta de varredura de aplicação, encaminhando o problema para o estado intermediário $\mathbf{Int_2}$. Finalizando o fluxo de execução do teste, em $\mathbf{Int_2}$ acontece a execução da ferramenta de exploração. Como resultado desta execução, obtém-se o relatório final do teste.

Utilizando as definições de planejamento em IA da Seção 2.4, o conjunto de estados S é definido neste problema como $S = \{E_i, Int_1, Int_2, E_f\}$.

4.2.2 Modelagem do problema de planejamento

A segunda etapa da atividade de planejamento do teste refere-se à modelagem em PDDL do problema definido na etapa anterior. As ações são modeladas conforme os tipos indicados no problema de planejamento em IA e representam as ferramentas e *framework* selecionados no estudo exploratório do Capítulo 3. A amostra de ferramentas selecionadas no estudo exploratório é composta pelas ferramentas Arachni, HTCAP, Skipfish, SQLmap, Wapiti, XSSer e ZAP e pelo *framework* Metasploit. Estas ferramentas selecionadas atendem ao fluxo de execução de um teste de intrusão. Consequentemente, nesta amostra, há ao menos uma ferramenta para cada tipo de ação definida em 4.2.1.

A Figura 4.3 apresenta as ações do problema de planejamento em IA associadas ao fluxo de execução das ferramentas de teste de intrusão. Como já mencionado, o teste se inicia com o testador indicando a URL da AST. A partir daí, existem fluxos de execução distintos para AST que não requer autenticação e AST que necessita de autenticação prévia para sua utilização. As ações elaboradas para representar configurações das ferramentas para execuções nestes dois tipos de ASTs são descritas a seguir. Os símbolos "●" e "○" indicam pontos de decisão. No entanto, os símbolos "●" se diferem pois também indicam o início de execução de uma ação.

- `Aut1`: representa a execução da ferramenta Wapiti, responsável por obter o identificador de sessão de ASTs que necessitam de autenticação;
- `Cr1`: representa a execução da ferramenta HTCAP como *crawler*. Esta ação indica a configuração da ferramenta para execuções em ASTs sem autenticação;
- `Cr2`: representa a execução da configuração da ferramenta HTCAP como *crawler* em ASTs com autenticação;
- `Varr1`: representa a execução da ferramenta Skipfish, utilizada como opção para varredura de aplicação. Esta ação refere-se à configuração da ferramenta para ASTs sem autenticação;
- `Varr2`: representa a execução da ferramenta Arachni como alternativa de ferramenta de varredura de aplicação. Esta ação indica a configuração desta ferramenta para ASTs sem autenticação;

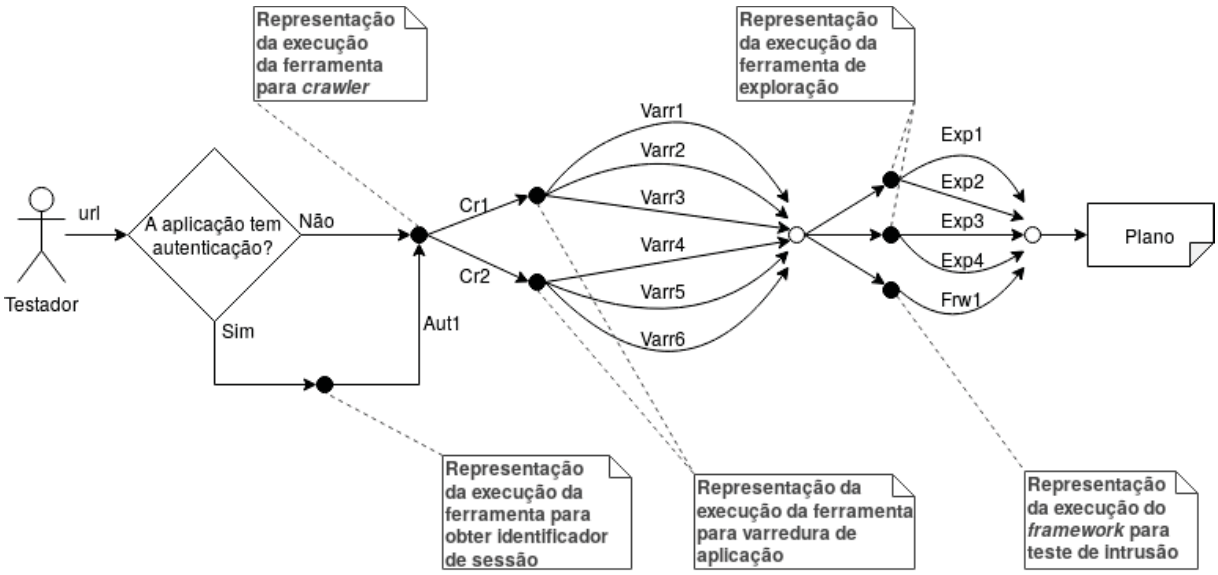


Figura 4.3: Ações associadas ao fluxo das ferramentas de teste de intrusão. Fonte: O autor (2019).

- `Varr3`: representa a execução da ferramenta ZAP como opção de ferramenta de varredura de aplicação. Esta ação representa a configuração utilizada em execuções com ASTs sem autenticação;
- `Varr4`: representa a execução da configuração da ferramenta Skipfish para varredura de ASTs com autenticação;
- `Varr5`: representa a execução da ferramenta Arachni configurada para testes em ASTs que necessitam de autenticação;
- `Varr6`: representa a execução da configuração da ferramenta ZAP em ASTs que requerem autenticação prévia;
- `Exp1`: representa a execução da ferramenta SQLmap para exploração da vulnerabilidade injeção de SQL em ASTs sem autenticação;
- `Exp2`: representa a execução da ferramenta XSSer para exploração da vulnerabilidade XSS. Esta ação representa a configuração da ferramenta para ASTs sem autenticação;
- `Exp3`: representa a execução da ferramenta SQLmap configurada para exploração de injeção de SQL em ASTs com autenticação;
- `Exp4`: representa a execução da ferramenta XSSer configurada para exploração de XSS em ASTs que requerem o processo de autenticação para sua utilização;
- `Frw1`: representa a execução de *exploits* pelo *framework* Metasploit como alternativa às ferramentas de exploração.

Conforme indicado na Figura 4.3, a escolha das ações ocorre conforme o tipo de AST. Para a escolha das ações que representam as ferramentas de exploração, além do tipo de AST, considera-se o tipo de vulnerabilidade que se deseja testar. Quanto à varredura, não observa-se características específicas para justificar a escolha entre as ferramentas da amostra. Desta forma, define-se uma prioridade referente à quantidade de vulnerabilidades detectadas por cada ferramenta de varredura utilizada.

A escala de prioridade definida contém valores de 1 a 3, sendo 1 a maior prioridade e 3 a menor prioridade. Por meio das análises realizadas no Capítulo 3, observa-se que a ferramenta ZAP detecta o maior número de vulnerabilidades entre as ferramentas de varredura selecionadas. Em seguida, aparecem as ferramentas Arachni e Skipfish. Assim, associa-se à ferramenta ZAP prioridade 1. Já para as ferramentas Arachni e Skipfish, associam-se as prioridades 2 e 3, respectivamente.

A atribuição das prioridades ocorre por meio de variáveis numéricas em cada ação declarada no arquivo do domínio em PDDL. As variáveis numéricas são posteriormente utilizadas como métricas nos arquivos de problema em PDDL. Para a geração do plano, o planejador minimiza o custo da função associada à variável numérica, resultando em um plano que contém a ação que representa a ferramenta com a maior prioridade.

Dadas as definições de estados e ações, a seguir é apresentada a modelagem do problema de planejamento em IA na linguagem PDDL. Esta modelagem é dividida em arquivos de domínio e problema, que compõem a representação de um problema de planejamento em IA na linguagem PDDL, conforme exemplificado na Subseção 2.4.2.

- **Modelagem do domínio em arquivo PDDL**

O domínio é modelado no arquivo `dominio_intrusao.pddl`, apresentado no Apêndice B. Este arquivo contém a modelagem de ações, conforme apresentado em 4.2.2, requisitos, tipos, predicados e função, descritos adiante. Além das descrições, são realizadas as associações destes itens que compõem o domínio em PDDL com as definições de planejamento em IA realizadas em 2.4.

- **Requisitos**: o arquivo de domínio contém a declaração de tipos e funções, logo os requisitos *:typing* e *:fluents* foram declarados.

- **Tipos**: os tipos declarados apresentados a seguir referem-se à AST, aos estados especificados no problema de planejamento em IA, às vulnerabilidades injeção de SQL e XSS e à execução de *exploits*.

```
aplicacao
estado
injecao_sql
xss
exploit
```

  Requisitos e tipos são componentes da linguagem PDDL. Logo, as declarações de ambos não constam na descrição formal de planejamento em IA.

- **Predicados**: os predicados são instanciados com objetos tipados de acordo com os tipos declarados. Os predicados desta modelagem são apresentados e descritos a seguir.

```
(estado ?s - estado)
(conexao_estados ?s1 ?s2 - estado)
(tem_autenticacao ?ast - aplicacao)
(autenticacao_obtida ?ast - aplicacao)
(crawler_executado ?ast - aplicacao)
```

```
(varredura_executada ?ast - aplicacao)
(exploracao_executada ?ast - aplicacao)
(teste_injecao_sql ?v - injecao_sql)
(teste_xss ?v - xss)
(executa_exploit ?e - exploit)
```

O domínio contém dois predicados referentes aos estados do problema: `estado`, utilizado para estabelecer a existência de um estado e `conexao_estados`, que estabelece a conexão existente entre dois estados. O predicado `tem_autenticacao` é definido para indicar a existência de autenticação na AST. Os predicados `autenticacao_obtida`, `crawler_executado`, `varredura_executada` e `exploracao_executada` são definidos para expressar que a execução das ferramentas para obtenção do identificador de sessão, *crawler*, varredura e exploração foram realizadas.

Os demais predicados especificados indicam o tipo de teste que se deseja realizar. O predicado `teste_injecao_sql` indica que o teste é realizado com o intuito de explorar a vulnerabilidade injeção de SQL na AST. O teste para exploração da vulnerabilidade XSS é indicada pelo predicado `teste_xss`. Já o predicado `executa_exploit` indica que um *exploit* deve ser executado pelo *framework* de teste de intrusão, o qual é utilizado como opção de ferramenta de exploração.

Utilizando as definições de planejamento em IA, $\boldsymbol{P}$, conjunto de predicados, é definido neste domínio como $\boldsymbol{P} = \{estado, conexao\_estados, tem\_autenticacao, autenticacao\_obtida, crawler\_executado, varredura\_executada, exploracao\_executada, teste\_injecao\_sql, teste\_xss, executa\_exploit\}$.

- **Função**: a função `prioridade` declarada é utilizada como variável numérica representando a prioridade de utilização das ferramentas de varredura de aplicação presentes na modelagem. A prioridade é indicada nas ações que representam a execução das ferramentas de varredura.

- **Ações**: as ações, especificadas pelos predicados e função definidos, correspondem às execuções de configurações das ferramentas conforme a aplicação que se deseja testar. A ação `Varr1`, apresentada a seguir, é utilizada como exemplo na descrição das ações, realizadas adiante.

```
;Varr1
(:action varredura_skipfish_sem_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado
                 ?proximo - estado)

  :precondition (and (estado ?atual)
                     (conexao_estados ?atual
                                      ?proximo)
                     (crawler_executado ?ast)
                     (not(varredura_executada ?ast))
                     (not(exploracao_executada ?ast))
```

```
                        (not(tem_autenticacao ?ast)))

    :effect         (and (estado ?proximo)
                         (not(estado ?atual))
                         (varredura_executada ?ast)
                         (increase (prioridade) 3))
)
```

Inicialmente, é associado um nome à ação, indicando a funcionalidade da ferramenta, o nome da ferramenta e a configuração necessária para o tipo de aplicação que se deseja testar. No exemplo, o nome `varredura_skipfish_sem_autenticacao` indica que a ação refere-se à execução da varredura com a ferramenta Skipfish configurada para ASTs sem autenticação.

Após o nome da ação, são indicados os parâmetros utilizados pelo ação. No exemplo, são passados como parâmetros a AST, o estado atual onde a ação é executada e do estado alcançado após sua execução. Nas ações referentes às ferramentas de exploração, a vulnerabilidade alvo da ferramenta também é passada como parâmetro.

Como pré-condição para a execução de uma ação, deve-se estar no estado atual e existir uma conexão entre os estados atual e seguinte, indicada pelo predicado `conexao_estados`. Ainda como pré-condições, são indicadas quais ferramentas devem ou não ter sido executadas até aquele momento. No exemplo, a ação representa a ferramenta de varredura, logo, é indicado pelo predicado `crawler_executado` que a ferramenta para *crawler* já deve ter sido executada. Finalizando as pré-condições, é indicada a existência de autenticação na AST. No exemplo, a negação do predicado `tem_autenticacao` indica que a ação refere-se à AST sem autenticação.

Como efeito da ação, ocorre a transição do estado `atual` para o estado `proximo`. Além disso, é feita a indicação que a execução da ferramenta que a ação representa foi realizada (no exemplo, indicada pelo predicado `varredura_executada`). Como efeito das ações que representam as execuções das ferramentas de varredura (`Varr1` a `Varr6`), é realizado o incremento da função `prioridade`, conforme escala de prioridade definida em 4.2.1.

Usando as definições de planejamento em IA, $\boldsymbol{A}$, conjunto de ações, é definido neste domínio como $\boldsymbol{A} = \{Aut1, Cr1, Cr2, Varr1, Varr2, Varr3, Varr4, Varr5, Varr6, Exp1, Exp2, Exp3, Exp4, Frw1\}$.

- **Modelagem do problema em arquivo PDDL**

Os arquivos de problema em PDDL são definidos para representar diferentes cenários de teste. Cada cenário de teste é especificado de acordo com o tipo de AST e a vulnerabilidade que se deseja testar. As ASTs se distinguem de acordo com a existência de autenticação para sua utilização. As possibilidades de teste referem-se às vulnerabilidades injeção de SQL, XSS e execução de *exploits*. O objetivo da criação de cenários distintos é exemplificar os possíveis fluxos de execução de teste nesta modelagem, conforme apresentado na Figura 4.3.

Além de informações sobre AST e vulnerabilidades, cada cenário contém a representação dos possíveis estados do problema, apresentados na Seção 4.2.1. Ou seja, contém a representação do conjunto $S = \{E_i, Int_1, Int_2, E_f\}$, com a indicação de como estes estados estão conectados

entre si, conforme apresentado na Figura 4.2. Ainda nos arquivos de problema, é feita a indicação da métrica que deve ser adotada pelo planejador para a geração dos planos.

A seguir são apresentados e descritos os arquivos de problema em PDDL definidos de acordo com cada cenário de teste. Além disso, são realizadas as associações dos componentes dos arquivos com as definições de planejamento em IA da Seção 2.4.

Cenário de teste 1 Modelado no arquivo `problema_cenario1.pddl`, representa o cenário de teste de exploração da vulnerabilidade injeção de SQL em AST que não requer autenticação. Este arquivo contém a definição dos objetos, estado inicial, estado final e métrica, descritos adiante.

```
;problema_cenario1.pddl
(define (problem problema_cenario1_injecao_sql)
  (:domain dominio_teste_intrusao)
  (:objects ast-sem-autenticacao - aplicacao
            injecao_sql - injecao_sql
            xss - xss
            exploit - exploit
            ei int1 int2 ef - estado)
  (:init
          (estado ei)
          (conexao_estados ei int1)
          (conexao_estados int1 int2)
          (conexao_estados int2 ef)
          (not (tem_autenticacao ast-sem-autenticacao))
          (teste_injecao_sql injecao_sql)
          (= (prioridade) 0))

  (:goal  (and (estado ef)))
  (:metric minimize (prioridade))
)
```

- **Objetos**: são declarados com um respectivo tipo, anteriormente declarado no arquivo de domínio, seguindo a sintaxe "`objeto - tipo`". Os objetos declarados são utilizados para instanciar os predicados, que por sua vez são utilizados para especificar os estados inicial e final. Este cenário contém os objetos apresentados a seguir.

  ```
  ast-sem-autenticacao
  injecao_sql
  xss
  exploit
  ei
  int1
  int2
  ef
  ```

  São declarados o objeto `ast-sem-autenticacao` do tipo `aplicacao`, os objetos `injecao_sql`, `xss` e `exploit` com seus respectivos tipos e os objetos `ei`, `int1`, `int2` e `ef` do tipo `estado`.

Ao menos um objeto de cada tipo declarado deve constar no arquivo de problema em PDDL a fim de se evitar conflitos durante a execução do planejador. Desta forma, alguns objetos declarados não são utilizados na modelagem do problema. Por exemplo, neste cenário os objetos `xss` e `exploit` são declarados, mas não são utilizados no decorrer do arquivo.

- **Estado inicial**: estabelece o estado `ei` como o estado inicial e indica as conexões existentes entre os estados (`ei` com `int1`, `int1` com `int2` e `int2` com `ef`). Estas informações repetem-se nos demais arquivos de problema apresentados nesta seção. Ainda no estado inicial, é feita a indicação que o teste se refere à detecção da vulnerabilidade injeção de SQL em AST sem autenticação. Finalizando a especificação do estado inicial deste cenário, a função `prioridade` é inicializada.

  De modo geral, conforme as definições de planejamento em IA, o estado inicial $\boldsymbol{I}$ é definido neste problema como $\boldsymbol{I} = \{E_i\}$.

- **Estado final**: de acordo com as definições de planejamento em IA, o estado final $\boldsymbol{O}$ é definido neste problema como $\boldsymbol{O} = \{E_f\}$. A definição de estado final repete-se nos demais cenários de teste descritos nesta seção.

- **Métrica**: a função `prioridade`, declarada no arquivo de domínio, é indicada como a métrica que o planejador deve minimizar durante a geração do plano.

Cenário de teste 2 Modelado no arquivo `problema_cenario2.pddl`, representa o cenário de teste de exploração da vulnerabilidade injeção de SQL em AST com autenticação. Objetos, estado inicial, estado final e métrica contidos neste arquivo são descritos a seguir.

```
;problema_cenario2.pddl
(define (problem problema_cenario2_injecao_sql)
  (:domain dominio_teste_intrusao)
  (:objects ast-com-autenticacao - aplicacao
            injecao_sql - injecao_sql
            xss - xss
            exploit - exploit
            ei int1 int2 ef - estado)
  (:init
          (estado ei)
          (conexao_estados ei int1)
          (conexao_estados int1 int2)
          (conexao_estados int2 ef)
          (tem_autenticacao ast-com-autenticacao)
          (teste_injecao_sql injecao_sql)
          (= (prioridade) 0))

  (:goal  (and (estado ef)))
  (:metric minimize (prioridade))
)
```

- **Objetos**: para este cenário, são declarados os seguintes objetos.

```
ast-com-autenticacao
injecao_sql
xss
exploit
ei
int1
int2
ef
```

  O objeto `ast-com-autenticacao` do tipo `aplicacao` difere esta declaração dos objetos do Cenário 1. De forma análoga ao Cenário 1, os objetos `xss` e `exploit` não são utilizados na descrição do problema.

- **Estado inicial**: de modo geral, é definido como $\boldsymbol{I} = \{E_i\}$, conforme definições de planejamento em IA. Além disso, define as conexões entre os estados, indica que a AST contém autenticação, especifica que o teste tem como objetivo explorar a vulnerabilidade injeção de SQL e inicializa a função `prioridade`.
- **Estado final**: é descrito, com as definições de planejamento em IA, como $\boldsymbol{O} = \{E_f\}$.
- **Métrica**: é indicada como métrica a função `prioridade`.

Cenário de teste 3 Este cenário de teste representa a exploração da vulnerabilidade XSS em aplicações que não contém autenticação. Este cenário é modelado no arquivo `problema_cenario3.pddl`, apresentado a seguir. A descrição dos objetos, estados inicial e final e métrica é realizada adiante.

```
;problema_cenario3.pddl
(define (problem problema_cenario3_xss)
  (:domain dominio_teste_intrusao)
  (:objects ast-sem-autenticacao - aplicacao
            injecao_sql - injecao_sql
            xss - xss
            exploit - exploit
            ei int1 int2 ef - estado)
  (:init
          (estado ei)
          (conexao_estados ei int1)
          (conexao_estados int1 int2)
          (conexao_estados int2 ef)
          (not (tem_autenticacao ast-sem-autenticacao))
          (teste_xss xss)
          (= (prioridade) 0))

  (:goal  (and (estado ef)))
  (:metric minimize (prioridade))
)
```

- **Objetos**: neste cenário são declarados os objetos a seguir.

```
ast-sem-autenticacao
injecao_sql
xss
exploit
ei
int1
int2
ef
```

  São declarados o objeto `ast-sem-autenticacao` do tipo `aplicacao`, além dos objetos referentes às vulnerabilidades e estados. Os objetos `injecao_sql` e `exploit` são declarados no arquivo, porém não constam na descrição do problema.

- **Estado inicial**: conforme as definições de planejamento em IA, é definido como $\boldsymbol{I} = \{E_i\}$. No estado inicial também são estabelecidas as conexões entre estados e indicado que o teste da vulnerabilidade XSS será realizado em aplicação sem autenticação.

- **Estado final**: com as definições de planejamento em IA, é descrito como $\boldsymbol{O} = \{E_f\}$.

- **Métrica**: a função `prioridade` é indicada com a métrica a ser utilizada para a geração do plano.

Cenário de teste 4 Este cenário de teste refere-se à exploração da vulnerabilidade XSS em aplicações que requerem autenticação. A seguir, é apresentado o arquivo `problema_cenario4.pddl`, que contém a modelagem deste cenário. Adiante, são descritos objetos, estado inicial, estado final e métrica contidos no arquivo.

```
;problema_cenario4.pddl
(define (problem problema_cenario4_xss)
  (:domain dominio_teste_intrusao)
  (:objects ast-com-autenticacao - aplicacao
            injecao_sql - injecao_sql
            xss - xss
            exploit - exploit
            ei int1 int2 ef - estado)
  (:init
          (estado ei)
          (conexao_estados ei int1)
          (conexao_estados int1 int2)
          (conexao_estados int2 ef)
          (tem_autenticacao ast-com-autenticacao)
          (teste_xss xss)
          (= (prioridade) 0))

  (:goal  (and (estado ef)))
  (:metric minimize (prioridade))
)
```

- **Objetos**: os objetos declarados neste cenário são apresentados a seguir.

```
ast-com-autenticacao
injecao_sql
xss
exploit
ei
int1
int2
ef
```

  Difere-se do Cenário 3 pela declaração do objeto `ast-com-autenticacao` do tipo `aplicacao`. Análogo ao Cenário 3, os objetos `injecao_sql` e `exploit` não são utilizados na descrição do problema, apesar de declarados no arquivo.

- **Estado inicial**: utilizando as definições de planejamento em IA, o estado inicial é definido $\boldsymbol{I} = \{E_i\}$. Também constam no estado inicial as conexões existentes entre os estados do problema, a indicação que a AST contém autenticação, a definição do teste da vulnerabilidade XSS e a inicialização da função `prioridade`.

- **Estado final**: $\boldsymbol{O} = \{E_f\}$, definido conforme as definições de planejamento em IA.

- **Métrica**: a função `prioridade` é definida como métrica para geração do plano.

Cenário de teste 5 Este cenário de teste representa a execução de *exploits* com um *framework* de teste de intrusão. Esta execução é realizada em casos onde não há uma ferramenta de exploração específica para a vulnerabilidade identificada na varredura. O cenário é modelado no arquivo `problema_cenario5.pddl`, apresentado a seguir. Os objetos, os estados inicial e final e a métrica contidos no arquivo são descritos adiante.

```
;problema_cenario5.pddl
(define (problem problema_cenario5_exploit)
  (:domain dominio_teste_intrusao)
  (:objects ast - aplicacao
            injecao_sql - injecao_sql
            xss - xss
            exploit - exploit
            ei int1 int2 ef - estado)
  (:init
          (estado ei)
          (conexao_estados ei int1)
          (conexao_estados int1 int2)
          (conexao_estados int2 ef)
          (executa_exploit exploit)
          (= (prioridade) 0))

  (:goal  (and (estado ef)))
  (:metric minimize (prioridade))
)
```

- **Objetos**: neste cenário são definidos os objetos apresentados a seguir.

```
ast
injecao_sql
xss
exploit
ei
int1
int2
ef
```

  Inicialmente, define-se o objeto `ast` do tipo `aplicacao`. Esta nomenclatura se difere dos objetos dos outros cenários de teste apresentados, pois a execução do *framework* não diferencia a AST de acordo com a existência de autenticação. Além deste objeto, são declarados objetos referentes às vulnerabilidades e aos estados. O objeto `exploit`, também declarado nos cenários anteriores, é utilizado apenas neste cenário. Os objetos `injecao_sql` e `xss` são declarados, porém não são utilizados na descrição subsequente do problema.

- **Estado inicial**: definido como $\boldsymbol{I} = \{E_i\}$, de acordo com as definições de planejamento em IA. Além disso, contém predicados que estabelecem a conexão entre os estados e indicam a execução do *exploit*.

- **Estado final**: conforme as definições de planejamento em IA, o estado final deste problema é definido como $\boldsymbol{O} = \{E_f\}$.

- **Métrica**: a função `prioridade` é definida com a métrica.

4.2.3 Geração dos planos de teste

Após a modelagem do problema na linguagem PDDL, ocorre a última etapa da atividade de planejamento do teste, que consiste na geração dos planos de teste com o planejador Metric-FF. Os planos de teste representam sequências de execução de ferramentas para identificar e explorar vulnerabilidades na AST. Cada cenário de teste apresentado em 4.2.2 origina planos de teste distintos. Para isso, utiliza-se como entradas do planejador o arquivo `dominio_intrusão.pddl` combinado com os arquivos de problema em PDDL que representam os cenários de teste definidos.

Os planos de teste gerados são constituídos por ações que representam execuções de ferramentas para obter identificador de sessão, *crawler*, varredura de aplicação e exploração de vulnerabilidades. Em relação à Figura 4.3, cada plano gerado sempre contém a ação `Varr1` e `Varr4`, que representam as varreduras com as duas configurações da ferramenta ZAP, e uma ação que representa cada ferramenta de exploração.

As configurações utilizadas em cada ferramenta se diferem de acordo com a existência do processo de autenticação na AST. Cada configuração é representada por ações distintas. A transição entre os estados do problema, definidos em 4.2.1, também é indicada em cada ação. Ou seja, as ações são apresentadas de acordo com a sintaxe "`configuração_da_ferramenta tipo_de_aplicação estado_atual próximo_estado`".

Utilizando as definições de planejamento em IA da Seção 2.4, um plano de teste para AST sem autenticação é definido, nesta modelagem, como $E_i \overset{Cr}{\rightarrow} Int_1 \overset{Varr}{\rightarrow} Int_2 \overset{Exp}{\rightarrow} E_f$. Ou seja, para se alcançar o estado final do problema, são necessárias ações que representam execuções de *crawler* (`Cr`), varredura de aplicação (`Varr`) e exploração (`Exp`). Já para AST com autenticação, o plano de teste é definido como $E_i \overset{Aut,Cr}{\rightarrow} Int_1 \overset{Varr}{\rightarrow} Int_2 \overset{Exp}{\rightarrow} E_f$. Nestes planos de teste, ocorre o acréscimo de uma ação `Aut`, que representa a execução da ferramenta para obtenção do identificador de sessão da AST.

A execução do planejador Metric-FF ocorre com sua configuração padrão, que determina como heurística de busca a aplicação de uma variação do algoritmo Subida de Encosta (EHC, do inglês *Enforced-Hill-Climbing*) seguida da realização de uma busca em largura (BFS, do inglês *Breadth-First Search*). A seguir, são descritos os planos de teste gerados pelo Metric-FF para cada cenário apresentado na Seção 4.2.2. Todos os planos gerados contêm custo `1`, que indica a prioridade associada à execução da ferramenta de varredura ZAP.

- **Plano para o Cenário 1**

A geração deste plano ocorre a partir do cenário de teste descrito no arquivo `problema_cenario1.pddl`. É constituído pelas ações `Cr1`, `Varr1` e `Exp1`, que representam a execução das ferramentas HTCAP, ZAP e SQLmap, respectivamente. A configuração selecionada de cada ferramenta refere-se à execução em AST sem autenticação. A seguir é apresentado o excerto do plano gerado pelo planejador Metric-FF.

```
step    0: CRAWLER_HTCAP_SEM_AUTENTICACAO
           AST-SEM-AUTENTICACAO EI INT1
        1: VARREDURA_ZAP_SEM_AUTENTICACAO
           AST-SEM-AUTENTICACAO INT1 INT2
        2: EXPLORACAO_SQLMAP_SEM_AUTENTICACAO
           INJECAO_SQL AST-SEM-AUTENTICACAO INT2 EF
plan cost: 1.000000
```

- **Plano para o Cenário 2**

Este plano é obtido com a execução do Metric-FF tendo como entrada o arquivo `problema_cenario2.pddl`. É composto pelas ações `Aut1`, `Cr2`,`Varr4` e `Exp3`, representando execução da ferramenta Wapiti, para obter as informações de autenticação da AST, e das ferramentas HTCAP, ZAP e SQLmap executadas com configurações para AST que requer autenticação. Um excerto do plano de teste contendo as ações é apresentado a seguir.

```
step    0: INFO_AUTENTICACAO_WAPITI
           AST-COM-AUTENTICACAO
        1: CRAWLER_HTCAP_COM_AUTENTICACAO
           AST-COM-AUTENTICACAO EI INT1
        2: VARREDURA_ZAP_COM_AUTENTICACAO
           AST-COM-AUTENTICACAO INT1 INT2
        3: EXPLORACAO_SQLMAP_COM_AUTENTICACAO
           INJECAO_SQL AST-COM-AUTENTICACAO INT2 EF
plan cost: 1.000000
```

- **Plano para o Cenário 3**

Este plano de teste é gerado a partir do cenário contido no arquivo `problema_cenario3.pddl`. As ações `Cr1`, `Varr1` e `Exp2` contidas no plano representam a execução das ferramentas HTCAP, ZAP e XSSer, respectivamente. Tais ações indicam configurações para AST que não requer autenticação para sua utilização. O excerto do plano de teste, obtido com o planejador Metric-FF, é apresentado a seguir.

```
step    0: CRAWLER_HTCAP_SEM_AUTENTICACAO
           AST-SEM-AUTENTICACAO EI INT1
        1: VARREDURA_ZAP_SEM_AUTENTICACAO
           AST-SEM-AUTENTICACAO INT1 INT2
        2: EXPLORACAO_XSSER_SEM_AUTENTICACAO
           XSS AST-SEM-AUTENTICACAO INT2 EF
plan cost: 1.000000
```

- **Plano para o Cenário 4**

Este plano é obtido com as informações do cenário de teste do arquivo `problema_cenario4.pddl`. As ações `Aut1`, `Cr2`, `Varr4` e `Exp4` que compõem este plano representam as execuções das ferramentas Wapiti, HTCAP, ZAP e XSSer configuradas para AST que necessita de autenticação para sua utilização. A seguir é apresentado o excerto do plano de teste resultante da execução do Metric-FF.

```
step    0: INFO_AUTENTICACAO_WAPITI AST-COM-AUTENTICACAO
        1: CRAWLER_HTCAP_COM_AUTENTICACAO
           AST-COM-AUTENTICACAO EI INT1
        2: VARREDURA_ZAP_COM_AUTENTICACAO
           AST-COM-AUTENTICACAO INT1 INT2
        3: EXPLORACAO_XSSER_COM_AUTENTICACAO
           XSS AST-COM-AUTENTICACAO INT2 EF
plan cost: 1.000000
```

- **Plano para o Cenário 5**

Este plano de teste é obtido a partir do arquivo `problema_cenario5.pddl`. Este plano contém três ações, que representam as execuções de ferramentas HTCAP para *crawler*, ZAP para varredura de aplicação e de execução de *exploits* com o *framework* Metasploit. Assim, com as definições de planejamento em IA, o plano de teste para este cenário é definido como $E_i \overset{Cr}{\rightarrow} Int_1 \overset{Varr}{\rightarrow} Int_2 \overset{Frw}{\rightarrow} E_f$, sendo `Frw` a ação que representa a execução do *framework*. Neste cenário, as execuções do *crawler* e varredura referem-se à configurações para aplicações sem autenticação. A seguir é apresentado o excerto do plano gerado com a execução do Metric-FF.

```
step    0: CRAWLER_HTCAP_SEM_AUTENTICACAO
           AUT EI INT1
        1: VARREDURA_ZAP_SEM_AUTENTICACAO
           AUT INT1 INT2
        2: EXPLORACAO_METASPLOIT_EXPLOIT
           EXPLOIT AUT INT2 EF
plan cost: 1.000000
```

4.3 EXECUÇÃO DO TESTE

A atividade de execução do teste inicia-se após a conclusão da atividade de planejamento do teste, onde planos de teste são obtidos como artefatos resultantes. Esta atividade consiste em executar testes de intrusão associando as informações contidas nos planos de teste gerados a partir da modelagem em PDDL com módulos de código automatizados.

A atividade de execução do teste no método de teste de intrusão proposto contém as etapas a seguir.

- **Instrumentalização do teste**: etapa para obtenção de informações da AST, como identificador de sessão e conjunto de páginas resultante da execução da ferramenta de *crawler*. Tais informações são utilizadas nas execuções posteriores das ferramentas de varredura e exploração. Esta etapa e o módulo elaborado para sua execução automática são apresentados na Subseção 4.3.1;
- **Execução da varredura de aplicação**: etapa para execução da ferramenta de varredura de aplicação. A descrição desta etapa e o módulo implementado para sua execução automática são apresentados na Subseção 4.3.2;
- **Execução da exploração de vulnerabilidades**: etapa para execução da ferramenta de exploração de vulnerabilidades identificadas na varredura. O módulo implementado para execução automática desta etapa é apresentado na Subseção 4.3.3;
- **Busca de *exploits***: etapa para busca de *exploits* e atualização da base de dados local do Metasploit, que se trata do *framework* incluído no método como alternativa às ferramentas de exploração. A descrição da proposta de módulo automatizável para a realização desta etapa é realizada na Subseção 4.3.4.

A Figura 4.4 apresenta o método de teste de intrusão, relacionando a atividade de execução do teste com o resultado da modelagem em PDDL. O resultado da modelagem dos arquivos em PDDL, apresentada na Subseção 4.2, é destacada em cinza representando os possíveis caminhos para o plano de teste. A indicação das entradas e saídas de cada execução realizada ao longo do método é apresentada conforme definições feitas em 2.3. São propostos módulos automatizados para a realização de cada etapa da atividade de execução do teste.

Conforme indicado na Figura 4.4, o método é composto de módulos para **M1**) execução das ferramentas de instrumentalização do teste, **M2**) execução da ferramenta de varredura de aplicação e **M3**) execução da ferramenta de exploração de vulnerabilidades. Estes módulos, adaptados para o estudo de caso do capítulo seguinte, são apresentados no Apêndice C. O método ainda contém uma proposta de módulo automatizável para **M4**) busca de *exploits* e atualização do *framework* Metasploit.

Para a execução do método, propõe-se um *script* adaptável conforme as informações dos planos de teste. A adaptação do *script* consiste em configurar a execução dos módulos **M1**, **M2** e **M3** de acordo com as indicações das ferramentas contidas nas ações do plano de teste. As perguntas referentes à autenticação na AST e às ferramentas de exploração de vulnerabilidades, indicadas na Figura 4.4 por losangos, são respondidas pelo testador ao longo do *script*.

As subseções seguintes descrevem os módulos contidos no método.

4.3.1 Instrumentalização do teste

A instrumentalização do teste é a primeira etapa da atividade de execução do teste do método proposto. A execução das ferramentas para instrumentalização do teste é representada

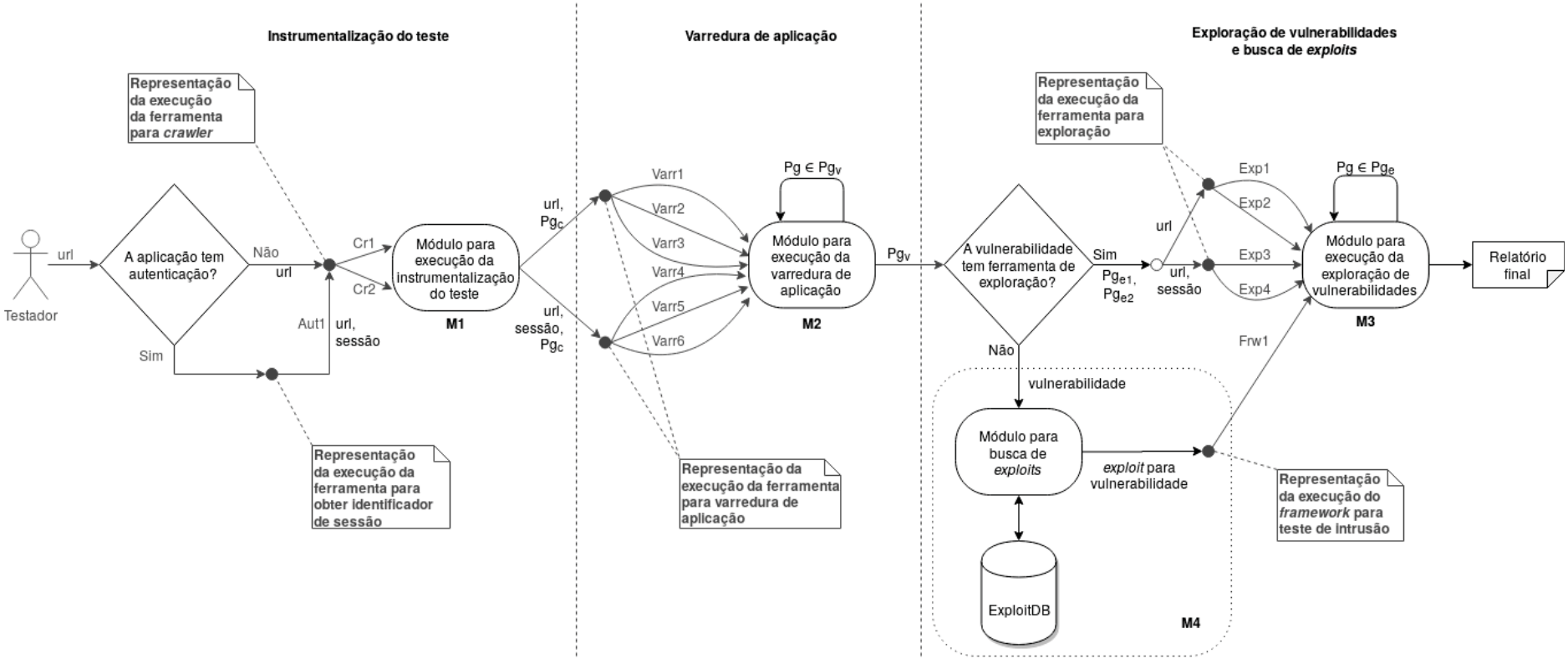


Figura 4.4: Execução do método de teste de intrusão usando o plano de teste. Fonte: O autor (2019).

pelo módulo **M1** contido no método. Inicialmente, o testador deve atribuir um usuário e uma senha para a autenticação das ASTs que requerem este processo. O testador deve incluir ao *script* de execução deste módulo as URLs de *login* e *logout* das ASTs que contém autenticação, além da página inicial de cada AST. Cabe ao testador incluir ao *script* o nome da AST e atribuir um valor *booleano* para indicar a existência de autenticação na aplicação.

Para testes de AST sem autenticação, o *crawler* é executado diretamente. Em contrapartida, quando a AST requer autenticação, se torna necessária a execução de uma ferramenta para obtenção do identificador de sessão, representada na Figura 4.4 pela ação `Aut1`. Após a obtenção do identificador de sessão, as execuções subsequentes de todas as ferramentas que compõem o método podem ocorrer com configurações distintas, de acordo com o tipo de AST.

Quando não há necessidade de autenticação na AST, utiliza-se como entrada da ferramenta de *crawler* apenas a URL da AST. A execução da ferramenta de *crawler* com esta configuração é representada pela ação `Cr1`. Já em casos onde a AST contém autenticação, são requeridos como entrada da ferramenta de *crawler* a URL e o identificador de sessão obtido com a execução de `Aut1`. Esta configuração é representada pela ação `Cr2`. A execução do *crawler* em ambas configurações resulta no conjunto Pg_c de páginas da AST, que se trata do dado de teste das execuções da ferramenta de varredura.

4.3.2 Execução da varredura de aplicação

Após a conclusão da instrumentalização do teste, inicia-se a etapa de execução da varredura de aplicação. Esta etapa é representada na Figura 4.4 pelo módulo **M2**. Este módulo é adaptado pelo testador no *script* de execução conforme a ferramenta selecionada no plano de teste. A execução das ferramentas de varredura são representadas pelas ações `Varr1` a `Varr6`. As ações `Varr1` a `Varr3` representam configurações referentes à AST sem autenticação. Já as ações `Varr4` a `Varr6` representam as configurações para AST com autenticação, então o identificador de sessão é encaminhado como entrada da ferramenta junto à URL da AST.

Utiliza-se como dado de teste em todas as configurações da ferramenta de varredura o conjunto Pg_c de páginas identificado pelo *crawler*. Cada página da AST contida no conjunto Pg_c é testada individualmente. Assim, são necessárias sucessivas execuções da ferramenta de varredura, utilizando as URLs destas páginas como dados de teste. Como resultado da execução deste módulo, obtém-se o conjunto Pg_v de páginas da AST com indicações de vulnerabilidades detectadas.

4.3.3 Execução da exploração de vulnerabilidades

Finalizando a atividade de execução do teste, ocorre a etapa de execução da ferramenta de exploração de vulnerabilidades. Esta etapa é representada pelo módulo **M3**, conforme apresentado na Figura 4.4. O conjunto Pg_v, obtido como resultado da ferramenta de varredura, é analisado pelo testador com o intuito de obter subconjuntos de páginas que contêm uma mesma vulnerabilidade. Cada um destes subconjuntos, representados por Pg_{e1} e Pg_{e2} na Figura 4.4, são utilizados como dados de teste de ferramentas de exploração distintas. As execuções destas ferramentas são representadas pelas ações `Exp1` a `Exp4`. Estas ações indicam configurações distintas de cada ferramenta de exploração, conforme o tipo de AST.

Análogo às ferramentas de varredura, as explorações das URLs do conjunto Pg_e ocorrem individualmente. As sucessivas execuções da ferramenta de exploração, tendo as URLs do conjunto Pg_e como dados de teste, são realizadas pela execução do módulo **M3**. O testador deve buscar informações no plano de teste sobre a AST e a vulnerabilidade a ser testada, permitindo

realizar as adaptações deste módulo no *script* de execução. A execução deste módulo resulta em um relatório contendo informações sobre a exploração realizada.

4.3.4 Busca de *exploits*

A proposta de módulo automatizável para busca de *exploits*, representado na Figura 4.4 pelo módulo **M4**, foi incluída no método para casos onde não há uma ferramenta de exploração específica para um conjunto Pg_e obtido. Nestes casos, sugere-se a realização de buscas de *exploits* para a vulnerabilidade presente nas páginas de um determinado conjunto Pg_e. Caso seja identificado, o *exploit* é adicionado à base local do Metasploit, que se trata do *framework* de teste de intrusão utilizado no método. Com a execução do Metasploit com este *exploit* identificado pretende-se a exploração da vulnerabilidade, gerando o relatório final do teste. A execução do Metasploit é representada pela ação `Fw1`.

Como etapa inicial do módulo M4 sugere-se a busca da vulnerabilidade identificada na varredura na lista Vulnerabilidades e Exposições Comuns (CVE, do inglês *Common Vulnerabilities and Exposures*) (CVE, 2019). CVE é uma lista de identificadores para vulnerabilidades conhecidas publicamente, mantida e atualizada pela comunidade de segurança computacional. O objetivo do CVE é padronizar identificadores de vulnerabilidades, proporcionando interoperabilidade entre diferentes bases de dados e ferramentas.

A seguir é apresentado um exemplo de uma vulnerabilidade no CVE. Inicialmente, é associado um número de identificação à vulnerabilidade, por exemplo `CVE-1999-0002`. Em seguida, é indicado o *status* da vulnerabilidade. O *status* `Entry`, como mostrado no exemplo, indica que a vulnerabilidade foi revisada e consta oficialmente na lista do CVE. O *status* `Candidates` indica que a vulnerabilidade ainda se encontra em revisão, podendo ser modificada e até mesmo rejeitada da lista.

Finalizando a descrição da vulnerabilidade no CVE, são apresentadas algumas referências pertinentes, como relatórios e uma breve descrição da vulnerabilidade. Existem Interfaces de Programação de Aplicação (APIs, do inglês *Application Programming Interfaces*) para a realização de buscas no CVE , como a API CVE-*Search*[1] e a API *Red Hat Security Data*[2]. Sugere-se que o testador realize a busca de *exploits* de vulnerabilidades com o *status* `Entry` na lista do CVE.

```
Name: CVE-1999-0002
Status: Entry
Reference: BID:121
Reference: URL:http://www.securityfocus.com/bid/121
Reference: CERT:CA-98.12.mountd
Reference: CIAC:J-006
Reference: URL:http://www.ciac.org/ciac/bulletins/j-006.shtml
Reference: SGI:19981006-01-I
Reference: URL:ftp://patches.sgi.com/support/free/security/
               advisories/19981006-01-I
Reference: XF:linux-mountd-bo
Buffer overflow in NFS mountd gives root access to remote
attackers, mostly in Linux systems.
```

[1]Disponível em: https://www.cve-search.org/api/

[2]Disponível em: https://access.redhat.com/documentation/en-us/red_hat_security_data_api

A base de dados de *exploits* sugerida para a busca neste módulo é a Exploit-DB (2019), por esta ser atualizada constantemente e disponibilizar *exploits* revisados no formato utilizado pelo *framework* Metasploit. Além disso, trata-se da base de dados de *exploits* mais utilizada pela comunidade de segurança computacional. A busca ocorre com a SearchSploit[3], uma ferramenta em linha de comando que realiza pesquisas por meio de uma cópia local do repositório do Exploit-DB. Antes da realização da busca com o SearchSploit, é feita uma atualização de seu repositório por meio do parâmetro "`-u`". Para a busca de *exploits* no repositório do SearchSploit, o testador indica o nome da vulnerabilidade ou o identificador da vulnerabilidade na lista CVE. Como resultado da busca, obtém-se o local onde se encontra o `arquivo.rb` que contém o *exploit* para a vulnerabilidade que se deseja explorar.

Após a identificação do *exploit* para a vulnerabilidade, este é adicionado à base de dados local do Metasploit. O `arquivo.rb` obtido é copiado para o diretório de *exploits* do Metasploit. Faz-se necessária a execução dos comandos `updatedb`, para atualização do banco de dados de nome de arquivos e `msfupdate`, para atualização da base local do Metasploit com o *exploit* encontrado no Exploit-DB. Após a atualização da base de dados do Metasploit, é possível sua execução, representada pela ação `Frw1` na Figura 4.4, para a exploração da vulnerabilidade, finalizando a execução do módulo e gerando o relatório final do teste.

Para exemplificar uma aplicação do módulo M4, são realizadas manualmente as etapas que compõem o método. A seguir, é apresentado um excerto da página inicial do Metasploit, onde é indicado que sua base local de *exploits* é composta inicialmente por 1947 *exploits*.

```
[ metasploit v5.0.60-dev-                          ]
[ 1947 exploits - 1089 auxiliary - 333 post        ]
[ 556 payloads - 45 encoders - 10 nops             ]
[ 7 evasion                                        ]
```

Antes da realização da busca, o repositório da ferramenta SearchSploit deve ser atualizado com o comando "`searchsploit -u`". Para a realização da busca neste exemplo, utilizam-se as palavras-chave "metasploit" e "linux", para indicar que se deseja *exploits* para o Metasploit no sistema operacional Linux. A seguir, é apresentado um excerto do resultado desta busca, com 3 dos 253 *exploits* identificados. Além dos nomes dos *exploits*, são indicados os caminhos onde os mesmos se encontram no repositório da ferramenta.

```
Título do Exploit  |  Caminho (/opt/exploitdb/)
-----------------------------------------------
ASX to MP3 converter 3.1.3.7 - '.asx' Local Stack Overflow
(Metasploit_ DEP Bypass)
| exploits/linux/local/47482.rb

AddressSanitizer (ASan) - SUID Executable Privilege Escalation
(Metasploit)
| exploits/linux/local/46241.rb

Adobe Flash Player - ActionScript Launch Command Execution
(Metasploit)
| exploits/linux/remote/18761.rb
```

[3]Disponível em: https://www.exploit-db.com/searchsploit

Um exemplo de atualização da base do Metasploit ocorre com o *exploit* "`Adobe Flash Player - ActionScript Launch Command Execution`". Este *exploit* é copiado de seu diretório de origem ("`/opt/exploitdb/exploits/linux/remote/18761.rb`") para o diretório de *exploits* do Metasploit ("`/opt/metasploit-framework/embedded/framework/modules/exploits`"). Após a inclusão deste *exploit*, a base local do Metasploit é atualizada, conforme excerto de sua página inicial apresentado a seguir.

```
[ metasploit v5.0.60-dev-                          ]
[ 1948 exploits - 1089 auxiliary - 333 post        ]
[ 556 payloads - 45 encoders - 10 nops             ]
[ 7 evasion                                        ]
```

Uma vez adicionado à base do Metasploit, o *exploit* é utilizado para os testes por meio do comando "`use`", conforme apresentado a seguir.

```
msf5 > use 18761.rb
msf5 exploit(18761) >
```

4.4 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou um método de teste de intrusão, dividido em atividades de planejamento do teste e execução do teste. A atividade de planejamento do teste do método, apresentada em 4.2, associa-se com a atividade de planejamento de teste de software, descrita em 2.1, cuja realização resulta em um artefato de plano de teste de software. A atividade de planejamento do teste no método contém etapas de definição do problema de planejamento em IA, modelagem do problema na linguagem PDDL e geração dos planos de teste. Esta atividade foi realizada com o objetivo de gerar planos de teste para diferentes cenários de teste, caracterizados pelo tipo de aplicação sob teste (AST) e vulnerabilidade que se deseja testar.

Inicialmente, foi realizada a etapa de definição do problema de planejamento em IA, apresentada em 4.2.1. Esta definição consistiu na caracterização dos estados e dos diferentes tipos de ações que compõem o problema. Para este problema de planejamento em IA, definiu-se que cada estado representa um momento distinto do teste de intrusão, compondo o conjunto $S = \{E_i, Int_1, Int_2, E_f\}$. Para a transição entre estados, considerou-se a execução de ações que representam as ferramentas presentes no fluxo de execução de um teste de intrusão.

Definiu-se que as ações representam as execuções das ferramentas para *crawler*, varredura de aplicação e exploração de vulnerabilidades. Para que o teste seja efetivo para AST com autenticação, percebeu-se a necessidade da inclusão de uma ação representando a execução de uma ferramenta que obtém o identificador de sessão. Optou-se também em incluir uma ação representando a execução de um *framework* para teste de intrusão, como alternativa em casos onde não há uma ferramenta de exploração para a vulnerabilidade encontrada na varredura.

Conforme as definições da linguagem PDDL, o problema de planejamento em IA foi modelado em arquivos de domínio e problema, apresentados em 4.2.2. O arquivo de domínio em PDDL contém as definições de ações e predicados. Para a definição das ações, foram consideradas as ferramentas selecionadas no Capítulo 3. Ao todo, foram definidas quatorze ações instanciadas por dez predicados distintos. Cada ação representa uma configuração específica de cada ferramenta selecionada. Uma configuração é composta por um conjunto de parâmetros utilizados na execução de uma ferramenta de acordo com o tipo de AST.

Os arquivos de problema em PDDL foram elaborados para exemplificar possíveis cenários de teste. Cada cenário foi especificado de acordo com o tipo de aplicação e vulnerabilidade a ser

testada. Desta maneira, foram elaborados cenários para aplicações com e sem autenticação e para testes das vulnerabilidades injeção de SQL e XSS. Considerou-se estas vulnerabilidades para a modelagem pois a amostra de ferramentas selecionada contém ferramentas para a exploração de ambas. Como alternativa a estes cenários, foi elaborado um cenário para execução de *exploits* com o *framework* Metasploit. Ao término da modelagem, obteve-se o problema de planejamento em IA para teste de intrusão descrito pela quádrupla (P, I, O, A), conforme definição de 2.4.

Cada arquivo de problema em PDDL representa uma instância para o arquivo de domínio em PDDL. Desta forma, cada cenário de teste, definidos nos arquivos de problema, ocasionou uma execução do planejador resultando na criação de planos de teste distintos. Para a geração dos planos de teste, conforme apresentado em 4.2.3, utilizou-se o planejador Metric-FF. Como métrica para geração dos planos, definiu-se uma escala de prioridade associada a cada ferramenta de varredura contida na modelagem. Desta forma, espera-se a escolha da ferramenta de varredura que realiza um teste mais abrangente, ou seja, detecta mais vulnerabilidades. Assim, como vantagem do uso do planejamento em IA no método tem-se a escolha de forma criteriosa de quais caminhos o teste deve percorrer.

Um plano de teste trata-se de um artefato construído estaticamente, indicando a sequência de ferramentas que o testador deve executar durante o teste. Além disso, cada ação contida no plano indica ao testador a necessidade de utilização de uma configuração específica de cada ferramenta. O plano de teste indica também a ferramenta de varredura que detecta mais vulnerabilidades entre as presentes na modelagem. Espera-se que estas características dos planos de teste sejam utilizadas como auxílio ao testador na especificação do teste e na realização da posterior atividade de execução do teste.

A atividade de execução do teste, apresentada em 4.3, é definida via plano de teste resultante da atividade de planejamento do teste. A instrumentalização do teste e execução das ferramentas de varredura e exploração ocorrem conforme informações fornecidas pelo plano de teste do cenário estabelecido. O método de teste de intrusão proposto associa o plano de teste estático com os módulos M1, M2 e M3 responsáveis pela execução automática das ferramentas indicadas no plano. Um *script* adaptável de acordo com o plano de teste foi proposto para a atividade de execução do teste do método. A execução das ferramentas utilizadas no método foi automatizada, logo não é mais custosa que os testes manuais realizados durante o estudo exploratório do Capítulo 3.

Como vantagem do método, destacou-se o uso de critérios para a geração do plano de teste, como o tipo de AST e o tipo de vulnerabilidade que se deseja testar. Desta forma, não se espera que o teste passe em todos os caminhos indicados na Figura 4.4, mas sim pelo caminho que indica as configurações necessárias das ferramentas para o teste de determinada AST. Durante a elaboração do método, percebeu-se que alguns de seus componentes não poderiam ser incluídos na modelagem em PDDL. Por exemplo, os módulos que representam as execuções das ferramentas de varredura de aplicação e exploração, cujas representações necessitam de laços e foram impossibilitadas pelas ações determinísticas, características da técnica de planejamento em IA utilizada nesta dissertação.

Além das ferramentas que compõem o fluxo de execução de um teste de intrusão, foi incluído no método o *framework* Metasploit. O Metasploit mostrou-se como uma alternativa às ferramentas de exploração, pois possui uma base local de *exploits* que pode ser atualizada pelo testador. A busca de *exploits* para vulnerabilidades sem uma ferramenta de exploração disponível e a atualização da base de *exploits* do Metasploit foi sugerida como um módulo automatizável no método. Sugeriu-se a utilização do Exploit-DB no método pois esta base de dados disponibiliza *exploits* verificados e atualizados constantemente pela comunidade de segurança computacional.

Além disso, o Exploit-DB disponibiliza *exploits* no padrão utilizado pelo Metasploit e oferece ferramental para busca automatizada de *exploits*.

Sugeriu-se a busca de vulnerabilidades no CVE como uma etapa anterior à busca de *exploits*. Existem APIs para busca de vulnerabilidades no CVE que requerem o identificador associado à vulnerabilidade ou a exata versão da vulnerabilidade. Tais informações não foram retornadas pela execução das ferramentas de varredura selecionadas para os testes desta dissertação. Além disso, a identificação de uma vulnerabilidade no CVE pode requerer uma busca exaustiva. Assim, a inclusão da busca automatizada de vulnerabilidades no CVE não foi incluída no método proposto. Este fato também impossibilitou a integração automatizada da ferramenta de varredura com o módulo de busca de *exploits*, cabendo ao testador identificar e indicar uma vulnerabilidade para esta etapa.

Como exemplo de execução do módulo M4, em 4.3.4 foi realizada uma busca de um *exploit* com a ferramenta SearchSploit e sua posterior inclusão na base local do *framework* Metasploit. Este exemplo, realizado manualmente, mostrou como ocorre a atualização da base do Metasploit com novos *exploits*. No entanto, em cenários reais o processo de identificação do *exploit* adequado para o teste de determinada vulnerabilidade se mostra uma tarefa não trivial, como já mencionado. Além disso, o *exploit* deve atender às características da aplicação em teste para o garantir a eficiência do teste. Em virtude destes apontamentos, propõe-se a automatização deste módulo como um dos trabalhos futuros decorrentes desta dissertação.

O capítulo seguinte apresenta um estudo de caso realizado para exemplificar uma aplicação do método de teste de intrusão proposto.

5 APLICAÇÃO DO MÉTODO DE TESTE DE INTRUSÃO

Este capítulo apresenta um estudo de caso realizado para exemplificar a aplicação do método de teste de intrusão proposto nesta dissertação. A Seção 5.1 apresenta a metodologia elaborada para a realização do estudo de caso. Inicialmente, é apresentada a motivação para a realização do estudo de caso, destacando objetivo e questão de pesquisa definidos. Em seguida, são descritos os objetos do estudo de caso, a configuração do ambiente de teste e as atividades para a realização do estudo de caso. Adiante, na Seção 5.2 são apresentados e analisados os resultados obtidos no estudo de caso. Finalizando o capítulo, são apresentadas considerações na Seção 5.3.

5.1 METODOLOGIA

O estudo de caso consiste na execução de testes de intrusão a partir dos planos de teste gerados com a modelagem em PDDL. A seguir, é apresentada a metodologia elaborada para a realização do estudo de caso que estabelece: a) o objetivo do estudo de caso; b) os objetos utilizados no estudo de caso; c) o ambiente configurado para a realização do estudo de caso; e d) atividades que compõem o estudo de caso.

a) Objetivo

O estudo de caso tem como objetivo exemplificar a execução dos módulos automatizados que compõem o método de teste de intrusão proposto em diferentes cenários de teste. A execução do método ocorre por meio da execução das ferramentas indicadas por um dos planos de teste gerados a partir da modelagem em PDDL apresentada nesta dissertação. Desta forma, pretende-se responder a seguinte questão de pesquisa com a realização deste estudo de caso.

- O plano de teste auxilia o testador para a definição e execução do teste de intrusão para um determinado cenário de teste?

b) Objetos

Os objetos do estudo de caso são ferramentas de teste de intrusão e aplicações sob teste (ASTs) apresentadas em 3.2. A amostra de ferramentas selecionadas para o estudo de caso é composta pelas ferramentas Wapiti, HTCAP, ZAP, SQLmap e XSSer. Já a amostra de ASTs selecionadas é composta pelas aplicações Badstore, BTSLab, bWAPP, DVWA, Mutillidae, XVWA e Webgoat. A biblioteca SQLite[1] é utilizada para o tratamento do conteúdo dos arquivos de saída.

c) Ambiente

O ambiente de teste do estudo de caso consiste em uma máquina virtual configurada nos servidores do Departamento de Informática (DInf) da UFPR. Os testes são executados em um computador Intel Core i5-2400 CPU @ 3.10GHz × 4, com 4.00GB de memória e sistema operacional Linux Mint 19.1 Cinnamon 4.0.10. As execuções dos testes ocorrem com as ASTs selecionadas configuradas em um ambiente *docker*, conforme descrito em 3.3.

[1]Disponível em: https://www.sqlite.org

d) Atividades

Este estudo de caso é baseado na realização de quatro atividades distintas: (i) definição do cenário de teste; (ii) adaptação do *script* de execução do teste; (iii) execução dos testes; e (iv) análise dos resultados obtidos com o teste. A sequência de atividades é apresentada na Figura 5.1. Todas as atividades são realizadas pelo testador.

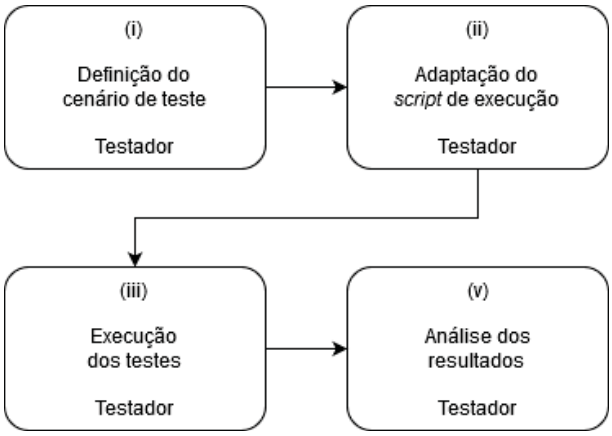


Figura 5.1: Metodologia do estudo de caso. Fonte: O autor (2019).

- Atividade (i): nesta atividade são definidos os cenários de teste que serão considerados para a aplicação do método.
- Atividade (ii): nesta atividade são feitas as adaptações no *script* de execução do teste. As adaptações consistem em incluir no *script* as configurações requeridas pelas ferramentas para a execução do teste de intrusão em cada tipo de AST. A configuração de cada ferramenta é indicada pelas ações do plano de teste gerado para os cenários de teste definidos na atividade (i).
- Atividade (iii): nesta atividade são realizadas as execuções dos testes com o *script* adaptado na atividade (ii).
- Atividade (iv): esta atividade consiste na análise do relatório resultante da realização da atividade (iii). Esta análise tem como objetivo observar se a exploração da vulnerabilidade na AST foi bem sucedida.

5.2 RESULTADOS E DISCUSSÃO

A seguir são apresentados e discutidos os resultados obtidos em cada atividade indicada pela metodologia elaborada para o estudo de caso.

- **Definição dos cenários de teste**

Conforme indicado na atividade (i) da metodologia, o estudo de caso inicia-se com a definição dos cenários de teste. Definiu-se para este estudo de caso a execução de testes das vulnerabilidades injeção de SQL e XSS. Desta forma, foram considerados os **Cenário de teste 1**, **Cenário de teste 2**, **Cenário de teste 3** e **Cenário de teste 4**, descritos em 4.2.2. A execução de testes referentes a estes cenários exemplifica a aplicação dos módulos M1, M2 e M3 do método de teste de intrusão proposto.

- **Adaptação dos *scripts* de execução**

Após a definição dos cenários de teste, os *scripts* de execução dos módulos foram adaptados, conforme estabelecido na atividade (ii) da metodologia. A adaptação dos *scripts* ocorreu de acordo com as indicações dos planos de teste de cada cenário, apresentados em 4.2.3. Como resultado das adaptações nestes módulos, foram obtidos os *scripts* apresentados no Apêndice C. A seguir, são descritas as adaptações realizadas em cada módulo do método.

- **Módulo M1**: para a adaptação do módulo M1, referente à instrumentalização do teste, foram incluídos ao *script* informações referentes às ASTs selecionadas para o estudo de caso. Foram incluídos os nomes das ASTs (por exemplo, "`Webgoat`"), as URLs de *login* das ASTs que contêm autenticação ("`http://testesseg.c3sl.ufpr.br:3001/WebGoat/login`", por exemplo), as URLs de *logout* que devem ser desconsideradas no teste (por exemplo, "`http://testesseg.c3sl.ufpr.br:3001/WebGoat/logout`") e as URLs das páginas iniciais das ASTs ("`http://testesseg.c3sl.ufpr.br:3001/WebGoat/start.mvc`", por exemplo). Além dessas informações, foram atribuídos às ASTs valores *booleanos*, indicando a existência do processo de autenticação. Por exemplo, atribuiu-se o valor "`1`" à AST Webgoat, pois essa aplicação contém autenticação.

  Ainda como parte da instrumentalização do teste, foi incluído ao *script* do módulo M1 a ferramenta Wapiti-getcookie, extensão da ferramenta Wapiti. A execução desta ferramenta gerou um arquivo no formato JSON contendo o identificador das ASTs que requerem autenticação. Foram incluídas também as duas configurações de execução da ferramenta HTCAP. Na configuração referente às ASTs com autenticação, utilizou-se como parte da entrada o arquivo JSON gerado pela ferramenta Wapiti-getcookie. A execução da ferramenta HTCAP, em ambas configurações, gerou um arquivo no formato db, contendo as URLs de páginas de cada AST. Para o tratamento das URLs contidas neste arquivo, utilizou-se uma funcionalidade da biblioteca SQLite. O módulo M1, adaptado para o estudo de caso, é apresentado em C.1.

- **Módulo M2**: para a varredura de aplicação do módulo M2, utilizou-se a ferramenta ZAP-cli, versão em linha de comando da ferramenta ZAP. Esta ferramenta utiliza como entrada o arquivo gerado pela ferramenta HTCAP. As URLs deste arquivo, referentes às páginas das ASTs, são testadas individualmente pela ferramenta. Como saída, foi gerado um relatório com indicações de vulnerabilidades detectadas nas URLs contidas no arquivo de entrada, além de alertas para possíveis vulnerabilidades. A execução desta ferramenta associa um número de identificação e uma escala de risco para cada vulnerabilidade identificada. A adaptação deste módulo para o estudo de caso é apresentada em C.2.

- **Módulo M3**: para a exploração de vulnerabilidades, o módulo M3 foi adaptado com as ferramentas SQLmap e XSSer. O arquivo resultante da execução da ferramenta ZAP foi analisado a fim de se obter as entradas para as ferramentas de exploração. Para esta análise, as informações deste arquivo foram convertidas em uma tabela e, conforme os valores contidos na coluna referente ao número de identificação da vulnerabilidade, a URL da AST foi encaminhada para a ferramenta de exploração adequada. O valor "`79`" é associado à vulnerabilidade XSS, logo as URLs com este valor são encaminhadas para a ferramenta XSSer. Já as URLs com valor "`89`" são testadas com a ferramenta SQLmap, pois contêm a indicação da vulnerabilidade injeção de SQL. O módulo M3, adaptado para o estudo de caso, é apresentado em C.3.

- **Execução dos testes e análise dos resultados**

Os resultados obtidos com a execução dos testes (atividade (iii) da metodologia) e as análises destes resultados (atividade (iv) da metodologia) são apresentados a seguir. Os arquivos resultantes das execuções das ferramentas realizadas ao longo do estudo de caso estão disponíveis no *link* aberto **https://github.com/lf-lima/RelatorioTesteIntrusao**.

Os módulos M1, M2 e M3 foram executados com o intuito de exemplificar, respectivamente, uma aplicação das etapas de instrumentalização de teste, varredura de aplicação e exploração de vulnerabilidades contidas no método proposto. Com a execução da ferramenta Wapiti-getcookie, no contida módulo M1, foram gerados três arquivos contendo os identificadores de sessão associados às ASTs bWAPP, DVWA e Webgoat. Cada um destes arquivos contêm a *string* `value` que representa o identificador de sessão para usuário e senha definidos anteriormente para cada aplicação. Ainda na execução do módulo M1, os arquivos com os identificadores de sessão das ASTs foram utilizados como parte da entrada da ferramenta HTCAP. A execução desta ferramenta como *crawler* resultou em arquivos contendo as URLs das ASTs que foram submetidas posteriormente à varredura. Com a análise destes arquivos resultantes, constata-se que a ferramenta HTCAP é capaz de lidar com o processo de autenticação, requerido pelas ASTs bWAPP, DVWA e Webgoat.

A Tabela 5.1 apresenta uma síntese dos resultados obtidos com as execuções do módulos M2 e M3 em cada AST selecionada para o estudo de caso. Na coluna "Varredura ZAP", é feita a indicação de vulnerabilidades identificadas nas ASTs e alertas emitidos durante a execução da varredura realizada com o módulo M2. Já a coluna "Exploração" indica se a execução das explorações com as ferramentas SQLmap e XSSer, referente à execução do módulo M3, foi bem sucedida. Os marcadores "✓" indicam vulnerabilidades e alertas identificados na AST pela varredura e exploração bem sucedida pelas ferramentas selecionadas, enquanto os marcadores "-" indicam resultados negativos para estes itens.

Tabela 5.1: Síntese dos resultados do estudo de caso. Fonte: O autor (2019).

Aplicação sob teste	**Varredura ZAP**			**Exploração**	
	Injeção de SQL	XSS	Alerta	SQLMap	XSSEr
Badstore	-	-	✓	-	-
BTSlab	-	✓	✓	-	-
bWAPP	-	-	✓	-	-
DVWA	-	-	✓	-	-
Mutillidae	✓	✓	✓	-	-
XVWA	-	✓	✓	-	-
Webgoat	-	-	✓	-	-

Os arquivos gerados pela ferramenta HTCAP foram utilizados como entrada da ferramenta ZAP-cli, contida no módulo M2. A execução desta ferramenta resulta em um arquivo txt, contendo vulnerabilidades e alertas identificados nas ASTs. Como indicado na Tabela 5.1, a vulnerabilidade injeção de SQL foi detectada apenas na AST Mutillidae, que corresponde a aproximadamente 14% da amostra de ASTs. Já a vulnerabilidade XSS foi identificada em aproximadamente 43% da amostra, sendo detectada nas ASTs BTSlab, Mutillidae e XVWA. Além destas vulnerabilidades, a execução da ferramenta ZAP-cli emitiu alertas que indicam a existência de outras possíveis vulnerabilidades em todas as ASTs da amostra.

Conforme a análise das ASTs realizada em 3.3, a identificação das vulnerabilidades injeção de SQL e XSS nas ASTs Mutillidae, BTSlab e XVWA se mostra consistente. Ainda de

acordo com a análise de 3.3, constata-se que as demais aplicações da amostra também contêm em suas implementações as vulnerabilidades injeção de SQL e XSS, conforme informações levantadas em suas documentações. No entanto, a execução da ferramenta ZAP-cli não identificou estas vulnerabilidades nas varredura das mesmas. Como indícios para estes resultados, têm-se possíveis limitações dos testes realizados com a versão da ferramenta ZAP-cli utilizada perante às implementações das vulnerabilidades nas ASTs.

As entradas para as ferramentas de exploração SQLmap e XSSer foram originadas a partir do arquivo gerado na execução da ferramenta ZAP. Analisando os relatórios resultantes da execução das SQLmap e XSSer, observa-se que não foi possível a exploração das vulnerabilidades injeção de SQL e XSS em nenhuma página das ASTs, conforme indicado na Tabela 3.3. Como possibilidades para este resultado, apontam-se as versões, tipos ou padrões destas vulnerabilidades utilizados na implementação das aplicações que podem não ser suportados pelas versões das ferramentas de exploração utilizadas.

5.3 CONSIDERAÇÕES DO CAPÍTULO

Este capítulo apresentou um estudo de caso para exemplificar a aplicação do método de teste de intrusão proposto. Os objetos do estudo de caso referem-se ao ferramental necessário para exemplificar o método em diferentes cenários de testes. A ferramenta Wapiti-getcookie, extensão da ferramenta Wapiti, foi utilizada para obter o identificador de sessão das ASTs que requerem autenticação. A ferramenta HTCAP foi definida como o *crawler* para as execuções. Para a varredura de aplicação, foi escolhida a ferramenta ZAP-cli, versão em linha de comando da ferramenta ZAP. As ferramentas SQLMap e XSSer foram selecionadas para a exploração das vulnerabilidades injeção de SQL e XSS, respectivamente.

As aplicações Badstore, BTSLab, bWAPP, DVWA, Mutillidae, XVWA e Webgoat foram definidas como as ASTs do estudo de caso. Estas aplicações contêm as vulnerabilidades injeção de SQL e XSS em seus conjuntos de vulnerabilidades V_{ast}, de acordo com a definição apresentada em 2.3. Isto permitiu a realização de testes de *benchmark* com as ferramentas selecionadas. Objetivou-se com a escolha destas ASTs a execução de testes com os dois tipos possíveis de ASTs presentes na modelagem em PDDL realizada nesta dissertação. Assim, foram realizados testes abrangendo todas as configurações de ferramentas, representadas por ações, presentes na modelagem.

A adaptação do *script* de execução dos módulos M1, M2 e M3 foi realizada conforme as indicações de configurações das ferramentas dos planos de teste de cada cenário definido. Portanto, respondendo a pergunta de pesquisa do estudo de caso, "o plano de teste auxilia o testador para a definição e execução do teste de intrusão para um determinado cenário de teste?", as indicações fornecidas pelos planos de teste mostraram-se condizentes com as configurações de ferramentas necessárias para a execução do teste em cada cenário definido para o estudo de caso. Assim, o plano de teste gerado mostrou sua utilidade como auxílio ao testador para a definição da atividade de execução do teste.

Conforme relatório obtido com a execução da ferramenta ZAP-cli, a integração das ferramentas se mostrou eficaz na identificação das vulnerabilidades injeção de SQL e XSS em algumas ASTs. Já a execução das ferramentas SQLmap e XSSer não resultou na exploração das vulnerabilidades nas ASTs, conforme relatórios resultantes destas ferramentas. Apontam-se como indícios para estes resultados das execuções das varreduras e explorações as tecnologias utilizadas pelas ASTs, versão e tipo das vulnerabilidades identificadas e possíveis limitações das ferramentas perante estas características das aplicações e vulnerabilidades.

6 CONSIDERAÇÕES FINAIS

Devido às especificidades das aplicações Web, a definição e a execução de testes de intrusão neste tipo de aplicação podem requerer grande esforço de tempo e recursos, além de expertise da equipe de teste. Esta dissertação apresentou um método automatizável de teste de intrusão para aplicações Web utilizando a técnica de planejamento em IA. O objetivo do método é gerar planos de teste para auxiliar os testadores no planejamento dos testes e disponibilizar módulos de código para a execução automatizada dos testes de intrusão.

Foram realizados dois mapeamentos sistemáticos da literatura para investigar a aplicação da técnica de planejamento em IA na engenharia de software. Com os mapeamentos, foi possível constatar a recente aplicação do planejamento em IA em teste de intrusão. Em uma busca direcionada a este tipo de teste, foi possível identificar na literatura duas abordagens distintas do uso do planejamento em IA. Identificou-se abordagens referentes à modelagem em PDDL de vulnerabilidades e de *exploits* pré-existentes como problemas de planejamento em IA.

Foi realizado um estudo exploratório com o intuito de analisar ferramentas de teste de intrusão. Este estudo consistiu na identificação de características das ferramentas, como funcionalidades e formatos de saída, além de execuções destas em aplicações Web vulneráveis. Durante este estudo, observou-se a necessidade de utilização de diferentes configurações na execução das ferramentas de acordo com o tipo de aplicação sob teste. Aplicações com autenticação, por exemplo, requerem execuções com parâmetros específicos para lidar com este tipo de processo. Assim, o planejamento do teste se mostra como uma atividade de auxílio aos testadores para a definição do ferramental necessário para a execução de um teste de intrusão.

Como resultado do estudo exploratório, definiu-se uma amostra de ferramentas de teste de intrusão utilizadas para a definição das ações que compõem a modelagem do problema de planejamento em IA presente no método proposto. Priorizou-se ferramentas de código aberto e executadas por linha de comando, visando a posterior integração e automatização do teste. Além disso, foram selecionadas ferramentas com funcionalidades necessárias para a execução do fluxo completo de um teste de intrusão e que atendam à metodologia PTES. Assim, a amostra foi composta pelas ferramentas Arachni, HTCAP, Skipfish, SQLmap, Wapiti, XSSer e ZAP. Além destas ferramentas, foi selecionado o *framework* Metasploit, utilizado como alternativa às ferramentas de exploração. A amostra contém as ferramentas de exploração SQLmap e XSSer, para as vulnerabilidades injeção de SQL e XSS, respectivamente. Assim, a modelagem com planejamento em IA e os testes realizados restringiram-se a estas vulnerabilidades.

Para o método de teste de intrusão proposto foram consideradas as atividades de planejamento do teste e execução do teste. A atividade de planejamento do teste consistiu em modelar a execução de um teste de intrusão com planejamento em IA culminando na geração de planos de teste. Nesta atividade, foram definidos os estados que representam o fluxo de execução de ferramentas de teste de intrusão. Após a definição dos estados, ocorreu a modelagem em arquivos de problema e domínio em PDDL. Durante esta etapa, a modelagem com PDDL se mostrou facilitada devido a sintaxe desta linguagem. Uma limitação observada durante a atividade de planejamento do teste refere-se ao tamanho da amostra de ferramentas, que restringiu a modelagem em PDDL a testes referentes às vulnerabilidades injeção de SQL e XSS.

Ao término da atividade de planejamento do teste, o planejamento em IA se mostrou eficaz para a definição criteriosa do plano de teste para um dado cenário. Isto foi notado com a utilização de critérios, como tipo de aplicação e vulnerabilidade, e de variáveis numéricas para a definição da ferramenta de varredura e representadas por funções da linguagem PDDL, que

proporcionaram a definição de uma métrica para minimização do custo na geração dos planos. No entanto, a utilização de uma técnica determinística de planejamento em IA se mostrou uma limitação pois impossibilitou a representação de alguns componentes do método, como os laços requeridos nas execuções das ferramentas de varredura e exploração. A atividade de execução do teste do método foi elaborada associando-se o resultado da modelagem em planejamento em IA com os módulos M1, M2 e M3, que automatizam a execução das ferramentas de teste de intrusão com a integração das ferramentas de instrumentalização do teste, varredura de aplicação e exploração de vulnerabilidades.

Um estudo de caso foi realizado com o intuito de exemplificar a aplicação do método proposto. Utilizou-se os planos de teste de diferentes cenários como artefatos estáticos como auxílio aos testadores para a execução dos testes. De acordo com as indicações de cada plano de teste, os *scripts* de execução foram adaptados, permitindo a execução do teste referente a cada cenário estabelecido. Ao término do estudo de caso, constatou-se que a sequência de configurações de ferramentas dada por cada plano de teste foi capaz de testar a vulnerabilidade de cada cenário de teste. Desta forma, em cenários de teste mais amplos, os planos de teste gerados pelo método podem auxiliar testadores sem expertise em teste de intrusão a definir o ferramental necessário de acordo com a aplicação sob teste e a vulnerabilidade que se pretende testar. A integração das ferramentas de teste de intrusão se mostrou consistente com a realização dos testes no estudo de caso. No entanto, algumas vulnerabilidades não foram detectadas e exploradas por possíveis limitações das ferramentas selecionadas para os testes.

A integração do módulo M4 de busca de *exploits* com as demais ferramentas não foi possível devido à falta de informações referentes às vulnerabilidades nas saídas geradas pelas ferramentas de varredura utilizadas. Assim, foi incluída no método uma atividade manual realizada pelo testador para análise dos relatórios de varredura, buscando identificar informações adicionais das vulnerabilidades, como versão e descrição na lista CVE. Apesar de não estar integrado à ferramenta de varredura, a metodologia estabelecida para este módulo se mostrou eficaz para busca de *exploits* para determinada vulnerabilidade e posterior atualização da base de dados local do Metasploit.

Portanto, esta dissertação se diferenciou das propostas encontradas na literatura por apresentar uma modelagem em PDDL da execução das ferramentas que compõem um teste de intrusão. No contexto de engenharia de software, foi notada a utilidade do método de teste de intrusão proposto quando se resulta em planos de teste que indicam ao testador as configurações necessárias das ferramentas para a execução dos testes com diferentes tipos de aplicação.

Além das contribuições citadas em 1.4, destaca-se como contribuição desta dissertação a definição de um modelo de configuração dado pelo plano de teste gerado com planejamento em IA que auxilia os testadores na de definição e posterior execução dos testes. Outra contribuição refere-se à integração das ferramentas que compõem um teste de intrusão, obtida com a automatização das execuções das ferramentas de instrumentalização de teste, varredura de aplicação e exploração de vulnerabilidades. Assim, considerando os planos de teste gerados e a integração das ferramentas realizada, aponta-se como contribuição do método facilitar a repetibilidade da execução dos testes de intrusão. Por fim, esta dissertação contribuiu com a proposta de um módulo automatizável de busca de *exploits* e atualização da base local do *framework* Metasploit.

6.1 TRABALHOS FUTUROS

Como trabalhos futuros decorrentes desta dissertação, sugerem-se:

- a inclusão das configurações das ferramentas na modelagem das ações em PDDL, permitindo a simulação do ambiente de execução do teste;
- a atribuição de outras características para as variáveis numéricas em PDDL, indicando o tempo de execução ou associando uma probabilidade à execução da ferramenta, por exemplo;
- a inclusão de ferramentas e vulnerabilidades na modelagem em PDDL, proporcionando a ampliação do plano de teste gerado;
- a replicação do estudo de caso realizado, considerando a utilização de outras ferramentas e testes para diferentes vulnerabilidades;
- a automatização do módulo M4, proporcionando sua integração com os módulos de instrumentalização do teste e varredura de aplicação; e
- a utilização de técnicas (Bercher e Mattmüller, 2009) e linguagem (Bertoli et al., 2003) de planejamento em IA não-determinístico, permitindo que ações que representam a execução de *exploits* sejam planejadas continuamente durante a construção do plano de teste, por exemplo.

REFERÊNCIAS

25010:2011, I. (2011). Systems and Software Engineering - Systems and Software Quality Requirements and Evaluation (SQuaRE) - System and Software Quality Models.

Albore, A., Palacios, H. e Geffner, H. (2009). A Translation-based Approach to Contingent Planning. Em *Twenty-First International Joint Conference on Artificial Intelligence*.

Amalio, N. (2009). Suspicion-Driven Formal Analysis of Security Requirements. *SECURWARE 2009*.

Andrews, M. e Whittaker, J. A. (2006). *How to Break Web Software: Functional and Security Testing of Web Applications and Web Services*. Addison-Wesley Professional.

Arachni (2019). Arachni: *Web Application Security Scanner Framework*. `https://www.arachni-scanner.com`. Acesso em: 1 ago. 2019.

BeEF (2019). *Browser Exploitation Framework* (beEF). `https://beefproject.com`. Acesso em: 1 ago. 2019.

Bercher, P. e Mattmüller, R. (2009). Solving non-deterministic planning problems with pattern database heuristics. Em *Annual Conference on Artificial Intelligence*, páginas 57–64. Springer.

Bertoli, P., Cimatti, A., Dal Lago, U. e Pistore, M. (2003). Extending pddl to nondeterminism, limited sensing and iterative conditional plans. Em *Proceedings of ICAPS'03 Workshop on PDDL*.

Bienvenu, M., Fritz, C. e McIlraith, S. A. (2006). Planning with Qualitative Temporal Preferences. *KR*, 6:134–144.

Blum, A. L. e Furst, M. L. (1997). Fast Planning Through Planning Graph Analysis. *Artificial intelligence*, 90(1-2):281–300.

Boddy, M. S., Gohde, J., Haigh, T. e Harp, S. A. (2005). Course of Action Generation for Cyber Security Using Classical Planning. Em *ICAPS*, páginas 12–21.

Bozic, J. e Wotawa, F. (2014). Security Testing Based on Attack Patterns. Em *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation Workshops*, páginas 4–11. IEEE.

Bozic, J. e Wotawa, F. (2015). PURITY: a Planning-based secURITY testing tool. Em *2015 IEEE International Conference on Software Quality, Reliability and Security-Companion*, páginas 46–55. IEEE.

Bozic, J. e Wotawa, F. (2017). Planning the Attack! Or How To Use AI In Security Testing? Em *IWAISe: First International Workshop on Artificial Intelligence in Security*, volume 50.

Bozic, J. e Wotawa, F. (2018). Planning-based Security Testing of Web Applications. Em *Proceedings of the 13th International Workshop on Automation of Software Test*, páginas 20–26. ACM.

Bozic, J. e Wotawa, F. (2019). Software Testing: According to Plan! Em *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, páginas 23–31. IEEE.

Clarke-Salt, J. (2009). *SQL Injection Attacks and Defense*. Elsevier.

Comer, D. (2016). *Interligação de Redes com TCP/IP–: Princípios, Protocolos e Arquitetura*, volume 1. Elsevier Brasil.

Crawler4j (2019). Crawler4j. `https://github.com/yasserg/crawler4j`. Acesso em: 1 ago. 2019.

CVE (2019). Common vulnerabilities and exposures (cve). `https://cve.mitre.org/`. Acesso em: 1 ago. 2019.

Czarnecki, K. e Kim, C. H. P. (2005). Cardinality-based Feature Modeling and Constraints: A Progress Report. Em *International Workshop on Software Factories*, páginas 16–20. ACM San Diego, California, USA.

decreasoner (2005). Discrete Event Calculus Reasoner - decreasoner. `http://decreasoner.sourceforge.net/`. Acesso em: 1 ago. 2018.

Delamaro, M., Jino, M. e Maldonado, J. (2007). *Introdução ao Teste de Software*. Elsevier Brasil, ISBN 978-8535226348.

Dierks, T. (2008). The Transport Layer Security (TLS) protocol - Version 1.2.

DS1 (2019). DS-1 Planner. `https://www.jpl.nasa.gov/missions/deep-space-1-ds1/`. Acesso em: 1 ago. 2019.

EL-Manzalawy, Y. (2006). Efficient Planning with Initial Irrelevant Facts.

Erol, K., Hendler, J. A. e Nau, D. S. (1994). UMCP: A Sound and Complete Procedure for Hierarchical Task-network Planning. Em *AIPS*, volume 94, páginas 249–254.

Exploit-DB (2019). Exploit-db: The Exploit Database. `https://www.exploit-db.com/`. Acesso em: 1 ago. 2019.

Feather, M. S. e Smith, B. (2001). Automatic Generation of Test Oracle from Pilot Studies to Application. *Automated Software Engineering*, 8(1):31–61.

Felderer, M., Büchler, M., Johns, M., Brucker, A. D., Breu, R. e Pretschner, A. (2016). Security Testing: A Survey. Em *Advances in Computers*, volume 101, páginas 1–51. Elsevier.

Fikes, R. E. e Nilsson, N. J. (1971). STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving. *Artificial intelligence*, 2(3-4):189–208.

Fowler, M., Scott, K. e Distilled, U. (1997). Applying the Standard Object Modeling Language. *England: Addison Wesley*.

Ghallab, M., Nau, D. e Traverso, P. (2004). *Automated Planning: Theory and Practice*. Elsevier.

Grant, T. (2018). Speeding up Planning of Cyber Attacks Using AI Techniques: State of the Art. Em *Proceedings, 13th International Conference on Cyber Warfare & Security (ICCWS), National Defense University, Washington DC*, páginas 235–244.

Grossman, J., Fogie, S., Hansen, R., Rager, A. e Petkov, P. D. (2007). *XSS Attacks: Cross Site Scripting Exploits and Defense*. Syngress.

Gupta, M., Fu, J., Bastani, F. B., Khan, L. R. e Yen, I.-L. (2007). Rapid Goal-oriented Automated Software Testing Using MEA-graph Planning. *Software Quality Journal*, 15(3):241–263.

Hoffmann, J. (2003). The Metric-FF Planning System: Translating "Ignoring Delete Lists" to Numeric State Variables. *Journal of artificial intelligence research*, 20:291–341.

Hoffmann, J. e Nebel, B. (2001). The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research*, 14:253–302.

Honiden, S., Nishimura, K., Uchihira, N. e Itoh, K. (1994). An Application of Artificial Intelligence to Object-oriented Performance Design for Real-time Systems. *IEEE transactions on software engineering*, (11):849–867.

Honiden, S., Uchihira, N. e Kasuya, T. (1985). *MENDEL: Prolog Based Concurrent Object Oriented Language*. Institute for New Generation Computer Technology.

Howe, A. E., von Mayrhauser, A. e Mraz, R. T. (1995). Test Sequences as Plans: An Experiment in Using an AI Planner to Generate System Tests. Em *Knowledge-Based Software Engineering Conference, 1995. Proceedings., 10th*, páginas 184–191. IEEE.

Howe, A. E., Von Mayrhauser, A. e Mraz, R. T. (1997). Test Case Generation as an AI Planning Problem. Em *Knowledge-Based Software Engineering*, páginas 77–106. Springer.

HTCAP (2019). HTCAP. `https://htcap.org`. Acesso em: 1 ago. 2019.

HttpClient (2019). Apache HttpComponents - HttpClient. `https://hc.apache.org/httpcomponents-client-ga/`. Acesso em: 1 ago. 2019.

IronWASP (2019). *Iron Web Application Advanced Security Testing Platform* (ironWASP). `https://resources.infosecinstitute.com/ironwasp-part-1-2`. Acesso em: 1 ago. 2019.

JavaGP (2017). JavaGP - Java Implementation of Graphplan. `https://github.com/pucrs-automated-planning/javagp`. Acesso em: 1 ago. 2019.

jsoup (2019). jsoup: Java HTML Parser. `https://jsoup.org/`. Acesso em: 1 ago. 2019.

Kniesel, G. (2006). *A Logic Foundation for Program Transformations*. Sekretariat für Forschungsberichte, Inst. für Informatik III.

Koehler, J., Nebel, B., Hoffmann, J. e Dimopoulos, Y. (1997). Extending Planning Graphs to an ADL Subset. Em *European Conference on Planning*, páginas 273–285. Springer.

Li, L., Wang, D., Shen, X. e Yang, M. (2009). A Method for Combinatorial Explosion Avoidance of AI Planner and the Application on Test Case Generation. Em *Computational Intelligence and Software Engineering, 2009. CiSE 2009. International Conference on*, páginas 1–4. IEEE.

Liaskos, S., McIlraith, S. A. e Mylopoulos, J. (2009). Towards Augmenting Requirements Models with Preferences. Em *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, páginas 565–569. IEEE Computer Society.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D. e Wilkins, D. (1998). PDDL- The Planning Domain Definition Language.

Memon, A. M., Pollack, M. E. e Soffa, M. L. (1999). Using a Goal-driven Approach to Generate Test Cases for GUIs. Em *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, páginas 257–266. IEEE.

Memon, A. M., Pollack, M. E. e Soffa, M. L. (2001). Hierarchical GUI Test Case Generation Using Automated Planning. *IEEE transactions on software engineering*, 27(2):144–155.

Metasploit (2019). Metasploit. `https://www.metasploit.com`. Acesso em: 1 ago. 2019.

Mraz, R. T., Howe, A. E., von Mayrhauser, A. e Li, L. (1995). System Testing With an AI Planner. Em *Software Reliability Engineering, 1995. Proceedings., Sixth International Symposium on*, páginas 96–105. IEEE.

Myers, G. J. (1979). The Art of Software Testing. *ISBN: 0-471-04328-1*.

Nau, D., Cao, Y., Lotem, A. e Munoz-Avila, H. (1999). SHOP: Simple Hierarchical Ordered Planner. Em *Proceedings of the 16th international joint conference on Artificial intelligence-Volume 2*, páginas 968–973. Morgan Kaufmann Publishers Inc.

Neto, D. N. (2019). Web (Eternamente) Revisitada: Análise de Vulnerabilidades Web e de Ferramentas de Código Aberto para Exploração. *Dissertação (Mestrado em Informática) - Programa de Pós-Graduação em Informática, Universidade Federal do Paraná.*

Newell, A., Shaw, J. C. e Simon, H. A. (1959). Report On A General Problem Solving Program. Em *IFIP congress*, volume 256, página 64. Pittsburgh, PA.

Obes, J. L., Sarraute, C. e Richarte, G. (2013). Attack Planning in the Real World. *arXiv preprint arXiv:1306.4044*.

OWASP (2017). Owasp Top 10 - 2017 The Ten Most Critical Web Application Security Risks. `https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project`. Acesso em: 1 out. 2019.

Pednault, E. P. (1989). ADL: Exploring the Middle Ground Between STRIPS and the Situation Calculus. *Kr*, 89:324–332.

Penberthy, J. S., Weld, D. S. et al. (1992). UCPOP: A Sound, Complete, Partial Order Planner for ADL. *Kr*, 92:103–114.

Pérez, J. e Crespo, Y. (2009). Perspectives on Automated Correction of Bad Smells. Em *Proceedings of the joint international and annual ERCIM workshops on Principles of software evolution (IWPSE) and software evolution (Evol) workshops*, páginas 99–108. ACM.

Petersen, K., Vakkalanka, S. e Kuzniarz, L. (2015). Guidelines for Conducting Systematic Mapping Studies in Software Engineering: An Update. *Information and Software Technology*, 64:1–18.

Pressman, R. (2011). *Engenharia de Software - 7ª Edição*. McGraw Hill. ISBN: 978-8563308-33-7.

PTES (2017). The Penetration Testing Execution Standard (PTES). http://www.pentest-standard.org. Acesso em: 1 ago. 2019.

Razavi, N., Farzan, A. e McIlraith, S. A. (2014). Generating Effective Tests for Concurrent Programs via AI Automated Planning Techniques. *International Journal on Software Tools for Technology Transfer*, 16(1):49–65.

Russell, S. J. e Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Malaysia; Pearson Education Limited,.

Sacerdoti, E. D. (1974). Planning in a Hierarchy of Abstraction Spaces. *Artificial intelligence*, 5(2):115–135.

Sarraute, C., Richarte, G. e Lucángeli Obes, J. (2011). An Algorithm to Find Optimal Attack Paths in Nondeterministic Scenarios. Em *Proceedings of the 4th ACM workshop on Security and artificial intelligence*, páginas 71–80. ACM.

Scheetz, M., von Mayrhauser, A. e France, R. (1999). Generating Test Cases from an OO Model with an AI Planning System. Em *Software Reliability Engineering, 1999. Proceedings. 10th International Symposium on*, páginas 250–259. IEEE.

Shmaryahu, D., Shani, G., Hoffmann, J. e Steinmetz, M. (2018). Simulated Penetration Testing as Contingent Planning. Em *Twenty-Eighth International Conference on Automated Planning and Scheduling*.

Skipfish (2019). Skipfish. https://tools.kali.org/web-applications/skipfish. Acesso em: 1 ago. 2019.

Soltani, S., Asadi, M., Gašević, D., Hatala, M. e Bagheri, E. (2012). Automated Planning for Feature Model Configuration Based on Functional and Non-functional Requirements. Em *Proceedings of the 16th International Software Product Line Conference-Volume 1*, páginas 56–65. ACM.

Soltani, S., Asadi, M., Hatala, M., Gasevic, D. e Bagheri, E. (2011). Automated Planning for Feature Model Configuration Based on Stakeholders Business Concerns. Em *Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering*, páginas 536–539. IEEE Computer Society.

Sommerville, I. (2012). Engenharia de software, 9 edição. *Pearson, Addison Wesley*, 8(9):10.

Sondik, E. J. (1978). The Optimal Control of Partially Observable Markov Processes Over the Infinite Horizon: Discounted Costs. *Operations research*, 26(2):282–304.

Sorrentino, F., Farzan, A. e Madhusudan, P. (2010). PENELOPE: Weaving Threads to Expose Atomicity Violations. Em *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, páginas 37–46. ACM.

SQLmap (2019). SQLmap. http://sqlmap.org. Acesso em: 1 ago. 2019.

Tunio, M. Z., Luo, H., Wang, C., Zhao, F., Shao, W. e Pathan, Z. H. (2018). Crowdsourcing Software Development: Task Assignment Using PDDL Artificial Intelligence Planning. *Journal of Information Processing Systems*, 14(1).

Vega (2019). Vega. https://subgraph.com/vega. Acesso em: 1 ago. 2019.

von Mayrhauser, A., Scheetz, M. e Dahlman, E. (1999). Generating Goal-Oriented Test Cases. Em *compsac*, página 110. IEEE.

von Mayrhauser, A., Scheetz, M., Dahlman, E. e Howe, A. E. (2000). Planner Based Error Recovery Testing. Em *Software Reliability Engineering, 2000. ISSRE 2000. Proceedings. 11th International Symposium on*, páginas 186–195. IEEE.

Von Mayrhauser, A., Walls, J. e Mraz, R. (1994). Sleuth: A Domain Based Testing Tool. Em *Proceedings., International Test Conference*, páginas 840–849. IEEE.

w3af (2019). *Web Application Attack and Audit Framework* (w3af). http://w3af.org. Acesso em: 1 ago. 2019.

Wapiti (2019). Wapiti. http://wapiti.sourceforge.net. Acesso em: 1 ago. 2019.

Wazlawick, R. (2013). *Engenharia de Software: Conceitos e Práticas*, volume 1. Elsevier Brasil.

Weidman, G. (2014). *Testes de Intrusão: Uma Introdução Prática ao Hacking*. Novatec, ISBN: 978-85-7522-558-5.

Wfuzz (2019). Wfuzz: *The Web Fuzzer*. https://wfuzz.readthedocs.io. Acesso em: 1 ago. 2019.

WordPad (2019). Microsoft WordPad. https://www.microsoft.com/en-us/p/wordpad-textnote/. Acesso em: 1 ago. 2019.

Wotawa, F. (2016). On the Automation of Security Testing. Em *Software Security and Assurance (ICSSA), 2016 International Conference on*, páginas 11–16. IEEE.

Wotawa, F. e Bozic, J. (2014). Plan it! Automated Security Testing Based on Planning. Em *IFIP International Conference on Testing Software and Systems*, páginas 48–62. Springer.

Xenotix (2019). Xenotix: XSS Exploit Framework. https://www.owasp.org/index.php/OWASP-Xenotix-XSS-Exploit-Framework. Acesso em: 1 ago. 2019.

XSSer (2019). Cross Site Scripting Framework (XSSer). https://www.owasp.org/index.php/Projects/OWASP_XSSER. Acesso em: 1 ago. 2019.

Yen, I.-L., Bastani, F. B., Mohamed, F., Ma, H. e Linn, J. (2002). Application of AI Planning Techniques to Automated Code Synthesis and Testing. Em *Tools with Artificial Intelligence, 2002.(ICTAI 2002). Proceedings. 14th IEEE International Conference on*, páginas 131–137. IEEE.

ZAP (2019). Zed Attack Proxy (ZAP). https://owasp.org/www-project-zap/. Acesso em: 1 ago. 2019.

APÊNDICE A – MAPEAMENTOS SISTEMÁTICOS DA LITERATURA

O protocolo elaborado para a realização dos mapeamentos sistemáticos da literatura (MSLs) apresentados adiante foi elaborado a partir da metodologia proposta por Petersen et al. (2015). O protocolo é composto por cinco atividades que consistem na definição do objetivo, das questões de pesquisa e das estratégias de busca, seleção de publicações e extração de dados. As atividades do protocolo são descritas a seguir.

1. **Definição do objetivo**: nesta atividade é definido o objetivo principal da realização do mapeamento sistemático;

2. **Definição das questões de pesquisa**: nesta atividade são elaboradas as questões de pesquisa que determinam o que deve ser respondido ao final do mapeamento. É elaborada uma questão de pesquisa principal e subquestões específicas de acordo com o tema a ser estudado;

3. **Definição do estratégia de busca**: esta atividade inicia-se com a definição das palavras-chave que devem representar o âmbito da pesquisa. Em seguida, são escolhidas as bases de dados que serão utilizadas para a realização da busca. Por fim, é realizada a elaboração das *strings* de busca de acordo com o padrão de cada base;

4. **Seleção de publicações**: esta atividade é realizada em duas etapas de filtragem de publicações. Na primeira etapa de filtragem é realizada uma seleção preliminar de publicações. As publicações selecionadas nesta etapa devem atender os critérios de inclusão pré-definidos. A seleção da amostra final de publicações é realizada na segunda etapa de filtragem com a aplicação de critérios exclusão;

5. **Extração de dados**: esta atividade inicia-se com a elaboração da ficha de extração de dados, que trata-se de um *template* que será utilizado em todas as publicações selecionadas. Os dados são então tabulados e agrupados para posterior análise.

A.1 PLANEJAMENTO EM IA EM REQUISITOS E DESIGN

Este mapeamento sistemático (**MSL-ReqDes**) tem como objetivo identificar a utilização da técnica de planejamento em inteligência artificial (IA) nas fases de desenvolvimento de software engenharia de requisitos (ER) e design. De acordo com este objetivo, a realização do mapeamento visa responder a seguinte questão de pesquisa (QP) principal.

- **QP** Existem trabalhos onde planejamento em IA é utilizado nas fases engenharia de requisitos e design?

  Foram também definidas outras nove subquestões, apresentadas e detalhadas a seguir.

- **QP1** Em qual fase de desenvolvimento de software o planejamento em IA é utilizado?

- **QP2** Qual a forma de utilização de planejamento em IA?

  Se ferramenta:

  - **QP2.1** Qual a ferramenta proposta?
  - **QP2.2** O código da ferramenta proposta é aberto?
- **QP3** Qual a técnica de planejamento em IA utilizada?
- **QP4** Qual(is) artefato(s) utilizado(s) para a definição do problema de planejamento?
- **QP5** Qual o planejador utilizado?
- **QP6** Qual(is) a(s) ferramenta(s) de apoio utilizada(s)?
- **QP7** O quê o plano gerado representa?

A QP1 tem como objetivo identificar a fase de desenvolvimento de software em que o planejamento em IA está sendo empregado. As possíveis respostas para esta questão são ER e design. A QP2 tem como objetivo identificar a forma que planejamento em IA foi utilizado nestas fases. As possíveis respostas são método, técnica, modelo, *framework* e ferramenta. Há duas subquestões em QP2 caso a forma utilizada seja ferramenta. Em QP2.1 é identificado o nome da ferramenta proposta e em QP2.2 é identificado se o código da ferramenta proposta é aberto.

A QP3 visa identificar a técnica de planejamento em IA utilizada pelas publicações. Como técnica, considerou-se teoria, características de implementação ou linguagem formal utilizada para a modelagem do problema de planejamento. Os artefatos utilizados como base para a definição do problema de planejamento são identificados na QP4. A QP5 identifica o planejador utilizado e a QP6 identifica as ferramentas de apoio empregadas. A QP7 identifica a representação do plano resultante.

Para a realização da busca, foram consideradas as seguintes palavras-chave: *"software engineering"* (em português, engenharia de software), *"requirements"* (em português, requisitos), *"design"*, *"AI planning"* (em português, planejamento em IA), e *"artificial intelligence planning"* (em português, planejamento em inteligência artificial). Optou-se por termos em inglês por se tratar do idioma adotado pela grande maioria das conferências e periódicos sobre o tema pesquisado.

As buscas foram realizadas nas bases de dados ACM[1], IEEE Xplore[2], Scopus[3] e Springer[4]. A Tabela A.1 apresenta as *strings* de busca utilizadas em cada base de dados e os filtros aplicados nas buscas. A aplicação dos filtros objetivou delimitar o escopo da pesquisa, abrangendo o maior número possível de publicações. Para isto, não foram aplicadas restrições com relação a tipo, local e ano de publicação. Foram selecionadas a área "ciência da computação" e disciplina "engenharia de software" nas bases que ofereciam estas opções.

Tabela A.1: MSL-ReqDes - *Strings* e filtros de busca por base de dados. Fonte: O autor (2019).

Bases de dados	***Strings* de busca**	**Filtros de busca**
ACM Digital	(+"*software engineering*" +(*requirements design*) +("*AI planning*" "*artificial intelligence planning*"))	*Select items from*: *All*; *Where*: *All*; *Published Since*: *All*

Continua na próxima página

[1]Disponível em: https://dl.acm.org/
[2]Disponível em: http://ieeexplore.ieee.org
[3]Disponível em: https://www.scopus.com
[4]Disponível em: https://link.springer.com

Tabela A.1 - *Continua da página anterior*

Bases de dados	***Strings* de busca**	**Filtros de busca**
IEEE	(“*software engineering*” AND (*requirements* OR *design*) AND (“*AI planning*” OR “*artificial intelligence planning*”))	*Search*: *All*; *Content filter*: *All*; *Publisher*: *All*; *Content Types*: *All*; *Publication Year*: *All years*
Scopus	(TITLE-ABS-KEY (“*software engineering*”) AND TITLE-ABS-KEY ((*requirements* OR *design*)) AND TITLE-ABS-KEY ((“*AI planning*” OR “*artificial intelligence planning*”))) AND (LIMIT-TO (SUBJAREA , “COMP”))	*Search*: *Article title*, *Abstract*, *Keywords*; *Range date*: *All*; *Document Type*: *All*; *Access Type*: *All*; *Subject Area*: *Computer Science*
SpringerLink	(*requirements* OR *design*) AND (“*AI planning*” OR “*artificial intelligence planning*”)	*Show documents published*: *All*; *Content type*: *All*; *Discipline*: *Computer Science*; *Subdiscipline*: *Software Engineering*, *Software Engineering/ Programming and Operating Systems*.

As buscas foram realizadas em outubro de 2018 no Departamento de Informática (DInf) da Universidade Federal do Paraná (UFPR) e retornaram 420 publicações. Para chegar na amostra final de publicações, foram realizadas duas etapas de filtragem. Durante a primeira etapa de filtragem, foi realizada a leitura do título, palavras-chave e resumo de todas as publicações retornadas na busca nas bases de dados. Foram então selecionadas publicações que atendem ao seguinte critério de inclusão (CI).

- **CI1** Publicações que utilizam planejamento em IA nas fases ER e design.

Com a aplicação de CI1, 17 publicações foram selecionadas, sendo 4 da ACM, 4 da IEEE Xplore, 6 da Scopus e 3 da Springer Link. A segunda etapa de filtragem ocorreu com estas 17 publicações que permaneceram na amostra. Nesta segunda etapa foram excluídas publicações de acordo com os critérios de exclusão (CE) a seguir.

- **CE1** Publicações que não estejam em inglês;
- **CE2** Publicações que apareceram de forma duplicada nas bases de dados;
- **CE3** Publicações que não estejam disponíveis integralmente;

- **CE4** Publicações referentes a livros e literatura cinzenta.

Com a aplicação da segunda filtragem, 10 publicações foram excluídas. Desta forma, a amostra final foi composta de 7 publicações, sendo 4 da ACM, 2 da IEEE Xplore e 1 da Scopus. Nenhuma publicação oriunda da busca na base Springer Link foi selecionada nesta etapa. A Tabela A.2 apresenta o número de publicações selecionadas por base de dados em cada fase de filtragem.

Tabela A.2: MSL-ReqDes - Número de publicações por bases de dados. Fonte: O autor (2019).

Base de dados	**Selecionados na busca inicial**	**Selecionados com aplicação de CI**	**Selecionados com aplicação de CE**
ACM	14	4	4
IEEE Xplore	307	4	2
Scopus	16	6	1
Springer Link	83	3	0
Total	420	17	10

As 7 publicações da amostra final foram submetidas à leitura completa e extração de dados. A extração de dados tem o intuito de responder as questões de pesquisa (QP) definidas para este MSL. Para isto, foi elaborada uma ficha de extração de dados, a fim de se de garantir que sejam aplicados os mesmos critérios de extração em cada publicação selecionada, facilitando deste modo a posterior análise e classificação das mesmas.

A ficha de extração de dados contém o dado a ser extraído, uma breve descrição do dado e a questão de pesquisa associada ao dado, conforme apresentado na Tabela A.3. Atribui-se a cada publicação selecionada um identificador (ID) e são indicados *link* (URL), título, autores e ano de publicação. Além disso, são identificados departamento e país de origem dos autores, informações sobre o local de publicação e resumo da proposta das publicações selecionadas. Estas informações não possuem uma questão de pesquisa associada, logo são indicados com o marcador "-". Os demais dados extraídos referem-se às QP estabelecidas para este MSL.

Tabela A.3: MSL-ReqDes - Ficha de extração de dados. Fonte: O autor (2019).

Dado extraído	**Descrição**	**Questão de pesquisa**
ID	Número de identificação	-
URL	Link para a publicação	-
Título	Título da publicação	-
Autores	Autores da publicação	-
Ano	Ano de publicação	-
Departamento	Departamento dos autores	-
País	País dos autores	-
Publicação	Informações sobre a publicação	-
Resumo	Resumo da proposta da publicação	-
Fase	Fase de desenvolvimento de aplicação	QP1

Continua na próxima página

Tabela A.3 - *Continua da página anterior*

Dado extraído	Descrição	Questão de pesquisa
Forma	Forma de utilização do planejamento em IA	QP2
Ferramenta	Ferramenta proposta pela publicação	QP2.1
Código ferramenta	O código da ferramenta proposta é aberto?	QP2.2
Técnica de planejamento	Técnica de planejamento em IA utilizada	QP3
Artefatos	Artefatos utilizados pelo planejamento em IA	QP4
Planejador	Planejador utilizado	QP5
Ferramenta de apoio	Ferramentas de apoio utilizadas	QP6
Plano	Plano resultante	QP7

A.1.1 Resultados

A Tabela A.4 apresenta um resumo da proposta de cada publicação selecionada. Adiante, são apresentados os resultados obtidos com a extração de dados na amostra final de 7 publicações. Os resultados são apresentados por questão de pesquisa (QP).

Tabela A.4: MSL-ReqDes - Resumo da proposta das publicações selecionadas. Fonte: O autor (2019).

Publicação	Proposta
Honiden et al. (1994)	Método para modelagem das fases de prototipação de sistemas em tempo real como um problema de planejamento
Amalio (2009)	Método para análise formal de requisitos de segurança utilizando o conceito de suspeita para guiar a busca de ameaças e vulnerabilidades em requisitos
Liaskos et al. (2009)	*Framework* para incluir a especificação de requisitos opcionais e preferências do usuário
Pérez e Crespo (2009)	Método para obter sequências de refatoração direcionadas para a correção de problemas na estrutura do sistema que podem afetar negativamente os fatores de qualidade do mesmo
Soltani et al. (2011)	*Framework* para a seleção automática de recursos adequados que atendem às preocupações de negócios das partes interessadas (do inglês, *stakeholders*) e limitações de recursos
Soltani et al. (2012)	*Framework* para seleção automática de recursos adequados que atendem às preferências (funcionais e não-funcionais) e restrições das partes interessadas

Continua na próxima página

Tabela A.4 - *Continua da página anterior*

Publicação	Proposta
Tunio et al. (2018)	Método para atribuição de tarefas no desenvolvimento de software em *crowdsourcing* através de atributos e personalidade do desenvolvedor

- **QP1: Em qual fase de desenvolvimento de software o planejamento em IA é utilizado?**

Para responder a QP1, as publicações selecionadas foram analisadas com o objetivo de identificar a fase de desenvolvimento de software onde as propostas são realizadas. Amalio (2009) e Liaskos et al. (2009) elaboram suas propostas na fase de engenharia de requisitos. Já Honiden et al. (1994), Pérez e Crespo (2009), Soltani et al. (2011), Soltani et al. (2012) e Tunio et al. (2018) desenvolvem suas propostas na fase de design.

Logo, dentre as publicações selecionadas,é possível notar a utilização mais frequente do planejamento em IA na fase de design (71% das publicações).

- **QP2: Qual a forma de utilização de planejamento em IA?**

Para a QP2, as publicações foram analisadas com o intuito de identificar a forma de utilização do planejamento em IA nas propostas. Liaskos et al. (2009), Soltani et al. (2011) e Soltani et al. (2012) elaboram propostas de *frameworks*. Honiden et al. (1994), Amalio (2009) e Tunio et al. (2018) apresentam propostas de métodos. Já Pérez e Crespo (2009) apresentam uma proposta de técnica.

As subquestões QP2.1 e QP2.2 não foram aplicadas durante a extração de dados, pois nenhuma das publicações selecionadas propõem ferramentas.

- **QP3: Qual a técnica de planejamento em IA utilizada?**

A QP3 foi respondida por meio da análise das publicações selecionadas para identificar as técnicas utilizadas referentes à teoria, implementação e linguagens de planejamento em IA. Foram identificadas três técnicas distintas nas publicações. A técnica Rede de Tarefa Hierárquica (HTN, do inglês *Hierarchical Task Network*) (Erol et al., 1994) é utilizada por Honiden et al. (1994), Pérez e Crespo (2009), Soltani et al. (2011) e Soltani et al. (2012). A técnica Linguagem de Preferência de Primeira Ordem (do inglês, *First-Order Preference Language*) (Bienvenu et al., 2006) é utilizada por Liaskos et al. (2009). A linguagem PDDL (McDermott et al., 1998) é utilizada por Tunio et al. (2018). Amalio (2009) não especifica a técnica de planejamento em IA utilizada.

- **QP4: Qual(is) artefato(s) utilizado(s) para a definição do problema de planejamento?**

Para a QP4, as publicações foram analisadas com o objetivo de identificar os artefatos utilizados para a definição do problema de planejamento em IA. Honiden et al. (1994) utilizam a descrição de objetos na linguagem orientada a objetos MENDEL (Honiden et al., 1985). Amalio (2009) utiliza o artefato de descrição de requisitos de segurança. Liaskos et al. (2009) utilizam artefato contendo a relação de preferências do usuário. Pérez e Crespo (2009) utilizam representações lógicas de programas em Java. Soltani et al. (2011) e Soltani et al. (2012) utilizam modelos de funcionalidades. Tunio et al. (2018) utilizam artefato de atributos da tarefa e do desenvolvedor.

- **QP5: Qual o planejador utilizado?**

Respondendo a QP5, foram identificados seis planejadores distintos nas publicações selecionadas. O planejador Abstrips (Sacerdoti, 1974) é utilizado por Honiden et al. (1994). O planejador *Discrete Event Calculus Reasoner* (decresoner) (decreasoner, 2005) é utilizado em Amalio (2009). Liaskos et al. (2009) utilizam o planejador PPLan (Bienvenu et al., 2006). Um planejador para a técnica HTN é identificado na proposta de Pérez e Crespo (2009). O planejador SHOP2 (Nau et al., 1999) é utilizado nas propostas de Soltani et al. (2011) e Soltani et al. (2012). Por fim, a proposta de Tunio et al. (2018) indica a utilização de um planejador para a linguagem PDDL.

- **QP6: Qual(is) a(s) ferramenta(s) de apoio utilizada(s)?**

Na QP6 foram identificadas as seguintes ferramentas de apoio nas publicações. Honiden et al. (1994) utilizam as ferramentas *Bottleneck Diagnosis Expert System* (BDES) e *Bottleneck Improvement Expert System* (BIES), BDES e BIES são ferramentas implementadas pelos autores da publicação para a análise qualitativa dos gargalos do sistema. Pérez e Crespo (2009) utilizam em sua proposta um conversor de representação lógica de programas Java (Kniesel, 2006). Soltani et al. (2011) e Soltani et al. (2012) utilizam a ferramenta *Visual-feature model plug-in* (vis-fmp) (Czarnecki e Kim, 2005), utilizada para descrever a configuração do problema (questões comerciais, custos e características do sistema). As demais publicações da amostra não especificam a utilização de ferramentas de apoio.

- **QP7: O quê o plano gerado representa?**

Para a QP7, foram realizadas análises dos planos gerados em cada publicação selecionada. O plano gerado por Honiden et al. (1994) representa um mecanismo para construção de protótipos usando componentes de software reutilizáveis. Na proposta de Amalio (2009), o plano representa uma possível ameaça de segurança ou vulnerabilidade. Em Liaskos et al. (2009), o plano estabelece uma sequência de ações que satisfaz as preferências do usuário. Em Pérez e Crespo (2009), o plano determina uma sequência de refatoração. O plano gerado pela proposta de Soltani et al. (2011) estabelece uma possível configuração para as funcionalidades baseadas nos requisitos das partes interessadas e preocupações comerciais. Já Soltani et al. (2012) estabelece um plano com uma seleção de funcionalidades que satisfazem requisitos funcionais e não-funcionais das partes interessadas. Por fim, Tunio et al. (2018) elabora um plano que determina atribuição das tarefas aos desenvolvedores.

A Tabela A.5 contém uma síntese dos resultados obtidos com a extração de dados.

Tabela A.5: MSL-ReqDes - Síntese da extração de dados. Fonte: O autor (2019).

Publicação	Fase	Proposta	Artefato	Técnica	Planejador	Apoio
Honiden et al., (1994)	Design	Método	Objetos em MENDEL	HTN	Abstrips	BDES e BIES
Amalio, (2009)	RE	Método	Requisitos de segurança	-	decresoner	-
Liaskos et al., (2009)	RE	*Framework*	Preferências do usuário	(*First-order Preference Language*)	PPLan	-

Continua na próxima página

Tabela A.5 - *Continua da página anterior*

Publicação	Fase	Proposta	Artefato	Técnica	Planejador	Apoio
Pérez and Crespo, (2009)	Design	Técnica	Representação de programas Java	HTN	(Planejador HTN)	Conversor Java
Soltani et al., (2011)	Design	*Framework*	Modelo de funcionalidades	HTN	SHOP2	vis-fmp
Soltani et al., (2012)	Design	*Framework*	Modelo de funcionalidades	HTN	SHOP2	vis-fmp
Tunio et al., (2018)	Design	Método	Tarefa e atributos do desenvolvedor	PDDL	(Planejador PDDL)	-

Os dados extraídos de acordo com as questões de pesquisa QP1 a QP6 são apresentados, respectivamente, nas colunas “Fase”, “Proposta”, “Artefato”, “Técnica”, “Planejador” e “Apoio” da Tabela A.5. Os dados entre parênteses não estavam contidos nas publicações e foram identificados por pesquisa adicional. Dados que não estavam presentes nas publicações são indicados pela marcação “-”.

A.2 PLANEJAMENTO EM IA EM TESTE DE SOFTWARE

Este mapeamento sistemático (**MSL-Teste**) tem como objetivo revisar o estado da arte das teorias, implementações, uso de métodos, técnicas e ferramentas de planejamento em IA aplicado a teste de software. Desta forma, a realização deste MSL visa responder a seguinte questão de pesquisa (QP) principal.

- **QP** Como o planejamento em IA é utilizado em teste de software?

Além desta questão de pesquisa principal, foram definidas oito subquestões com o objetivo de responder especificidades acerca do assunto estudado. Estas subquestões são apresentadas e detalhadas a seguir.

- **QP1** Quais são as formas de utilização de planejamento em IA no teste de software?

  Se ferramenta:

  - **QP1.1** Qual a ferramenta proposta?
  - **QP1.2** O código da ferramenta proposta é aberto?

- **QP2** Qual a fase de teste de software utilizada?
- **QP3** Qual a técnica de teste de software utilizada?
- **QP4** Qual a técnica de planejamento em IA utilizada?

- **QP5** Qual(is) artefato(s) utilizado(s) para definição do problema de planejamento?
- **QP6** Qual o planejador utilizado?
- **QP7** Qual(is) a(s) ferramenta(s) de apoio utilizada(s)?
- **QP8** O quê o plano gerado representa?

A QP1 tem como objetivo identificar a forma de utilização do planejamento em IA em teste de software. Dentre as possibilidades estão método, técnica, modelo, *framework* ou ferramenta. A QP1 possui duas subquestões aplicadas em caso da forma utilizada pela publicação ser ferramenta. Na QP1.1 é identificado o nome da ferramenta proposta e na QP1.2 é identificado se o código da ferramenta proposta é aberto.

Em QP2 e QP3 são identificadas informações relacionadas às fases e técnicas de teste de software. Na QP2, é identificada a fase de teste, podendo ser categorizada em teste de unidade, integração, sistema ou validação. Na QP3, é identificada a técnica de teste empregada, cujas possibilidades são teste funcional, estrutural ou baseado em erros. A QP4 visa identificar a técnica de planejamento em IA utilizada pela proposta. Os artefatos utilizados como base para a definição do problema de planejamento são identificados na QP5. A QP6 identifica a ferramenta de planejamento utilizada e a QP7 identifica as possíveis ferramentas de apoio utilizadas pelos autores. A QP8 analisa a representação do plano gerado.

Para a realização das buscas foram utilizados termos em inglês, por se tratar do idioma adotado pela maior parte de periódicos e conferência relevantes na área. Desta forma, foram definidas as seguintes palavras-chave: *"software engineering"* (em português, engenharia de software), *"software test"* (em português, teste de software), *"software testing"* (em português, teste de software), *"AI planning"* (em português, planejamento em IA), e *"artificial intelligence planning"* (em português, planejamento em inteligência artificial).

A Tabela A.6 apresenta as *strings* e filtros de busca utilizados em cada base de dados. Análogo ao MSL apresentado em A.1, as buscas aconteceram nas bases de dados ACM, IEEE, Scopus, e Springer, sem restrições quanto a tipo, local e ano de publicação. As opções "ciência da computação" e "engenharia de software" foram selecionadas quando disponibilizadas.

Tabela A.6: MSL-Teste - *Strings* e filtros de busca por base de dados. Fonte: O autor (2019).

Bases de dados	***Strings* de busca**	**Filtros de busca**
ACM Digital	(+"*software engineering*" +("*software test*" "*software testing*") +("*AI planning*" "*artificial intelligence planning*"))	*Select items from*: *All*; *Where*: *All*; *Published Since*: *All*
IEEE	("*software engineering*" AND ("*software test*" OR "*software testing*") AND ("*AI planning*" OR "*artificial intelligence planning*"))	*Search: All*; *Content filter*: *All*; *Publisher*: *All*; *Content Types*: *All*; *Publication Year*: *All years*

Continua na próxima página

Tabela A.6 - *Continua da página anterior*

Bases de dados	***Strings* de busca**	**Filtros de busca**
Scopus	(TITLE-ABS-KEY (“*software engineering*”) AND TITLE-ABS-KEY ((“*software test*” OR “*software testing*”)) AND TITLE-ABS-KEY ((“*AI planning*” OR “*artificial intelligence planning*”))) AND (LIMIT-TO (SUBJAREA , “COMP”))	*Search*: *Article title*, *Abstract*, *Keywords*; *Range date*: *All*; *Document Type*: *All*; *Access Type*: *All*; *Subject Area*: *Computer Science*
SpringerLink	(“*software test*” OR “*software testing*”) AND (“*AI planning*” OR “*artificial intelligence planning*”)	*Show documents published*: *All*; *Content type*: *All*; *Discipline*: *Computer Science*; *Subdiscipline*: *Software Engineering*, *Software Engineering/ Programming and Operating Systems*.

As buscas aconteceram em outubro de 2018 no Departamento de Informática (DInf) da UFPR, resultando em 149 publicações. Estas publicações foram então submetidas a duas etapas de filtragem. Para a primeira filtragem foi realizada a leitura do título, palavras-chave e resumo das publicações desta primeira amostra. As publicações que atendem ao critério de inclusão (CI) apresentado a seguir foram selecionadas.

- **CI1** Publicações que utilizam planejamento em IA em teste de software.

Após a aplicação de CI1, 15 publicações foram selecionadas, sendo 1 da ACM, 10 do IEEE, 2 da Scopus e 2 da SpringerLink. Estas publicações foram então submetidas à segunda etapa de filtragem, que consiste na aplicação dos critérios de exclusão (CE) apresentados a seguir.

- **CE1** Publicações que não estejam em inglês;
- **CE2** Publicações que apareceram de forma duplicada nas bases de dados;
- **CE3** Publicações que não estejam disponíveis integralmente;
- **CE4** Publicações referentes a livros e literatura cinzenta.

A aplicação dos CEs não resultaram na eliminação de nenhuma publicação. Desta forma, a amostra final manteve-se com 15 publicações. A Tabela A.7 apresenta o número de publicações em cada etapa de filtragem realizada.

As 15 publicações contidas na amostra final foram submetidas à leitura completa para extração de dados. Para isso, foi elaborada uma ficha de extração de dados, apresentada na Tabela A.8. Inicialmente, foram atribuídos identificadores (ID) a cada publicação selecionada. Foram

Tabela A.7: MSL-Teste - Número de publicações por bases de dados. Fonte: O autor (2019).

Base de dados	**Selecionados na busca inicial**	**Selecionados com aplicação de CI**	**Selecionados com aplicação de CE**
ACM	4	1	1
IEEE Xplore	136	10	10
Scopus	2	2	2
Springer Link	7	2	2
Total	149	15	15

então catalogados o *link* para o artigo (URL), título, autores, ano de publicação, departamento e país dos autores. Foram obtidas também informações do local de publicação e sínteses das propostas das publicações. Estas informações não possuem uma QP associada, logo são indicadas com o marcador "-". As demais informações referem-se às QP descritas no protocolo do MSL.

Tabela A.8: MSL-Teste - Síntese da extração de dados. Fonte: O autor (2019).

Dado extraído	**Descrição**	**Questão de pesquisa**
ID	Número de identificação	-
URL	Link para o artigo	-
Título	Título do artigo	-
Autores	Autores do artigo	-
Ano	Ano de publicação	-
Departamento	Departamento dos autores	-
País	País dos autores	-
Publicação	Informações sobre a publicação	-
Resumo	Resumo da proposta da publicação	-
Forma	Forma de utilização do planejamento em IA	QP1
Ferramenta	Ferramenta proposta pela publicação	QP1.1
Código ferramenta	O código da ferramenta proposta é aberto?	QP1.2
Fase de teste	Fase de teste de software utilizada	QP2
Técnica de teste	Técnica de teste de software utilizada	QP3
Técnica de planejamento	Técnica de planejamento em IA utilizada	QP4
Artefatos	Artefatos utilizados pelo planejamento em IA	QP5
Planejador	Planejador utilizado	QP6
Ferramenta de apoio	Ferramentas de apoio utilizadas	QP7
Plano	Plano resultante	QP8

A.2.1 Resultados

A Tabela A.9 apresenta uma síntese da proposta de cada publicação selecionada.

Tabela A.9: MSL-Teste - Resumo da proposta das publicações selecionadas. Fonte: O autor (2019).

Publicação	**Proposta**
Mraz et al. (1995)	Modelo para geração de dados de teste para comandos de linguagem da *StorageTek Robot Tape Library*
Howe et al. (1995)	Modelo para geração de dados de teste para a *StorageTek Robot Tape Library*
Howe et al. (1997)	Modelo para a geração de casos de teste para a *StorageTek Robot Tape Library*
Scheetz et al. (1999)	Modelo para geração de casos de teste que satisfazem os objetivos de testes derivados da UML
von Mayrhauser et al. (1999)	Modelo para geração de casos de teste com base em objetivos de teste de alto nível
Memon et al. (1999)	Técnica para geração automática de casos de teste para interface gráfica de usuário
von Mayrhauser et al. (2000)	Técnica para geração de teste de recuperação de erros com conceitos de teste de mutação
Feather e Smith (2001)	*Framework* para geração automática de testes *oracle*
Memon et al. (2001)	Ferramenta para geração automática de casos de teste para interface gráfica de usuário
Yen et al. (2002)	*Framework* para auxílio na síntese de código para desenvolver um sistema a partir de componentes existentes, bem como testes automatizados do sistema
Gupta et al. (2007)	Ferramenta para geração de dados de teste utilizando o algoritmo *Graphplan*
Li et al. (2009)	Ferramenta para geração de casos de teste utilizando múltiplos arquivos de entrada para ganho de desempenho
Razavi et al. (2014)	*Framework* para prever execuções simultâneas de programas que contêm condições de corrida, violações de atomicidade ou referências de ponteiros nulos.
Wotawa (2016)	Técnica para detecção de vulnerabilidades no sistema
Bozic e Wotawa (2018)	*Framework* para modelagem de vunerabilidades e teste de intrusão em aplicações Web

Os resultados obtidos com a extração de dados nas publicações da amostra final são apresentados a seguir por questão de pesquisa.

- **QP1: Quais são as formas de utilização de planejamento em IA no teste de software?**

Para responder a QP1, as publicações foram analisadas com o objetivo de identificar como o planejamento em IA foi utilizado nas propostas. Constata-se que as formas utilizadas são *framework* em Feather e Smith (2001), Yen et al. (2002), Razavi et al. (2014) e Bozic e Wotawa (2018); ferramenta em Memon et al. (2001), Gupta et al. (2007) e Li et al. (2009); modelo em

Mraz et al. (1995), Howe et al. (1995), Howe et al. (1997), Scheetz et al. (1999) e von Mayrhauser et al. (1999); e técnica em Memon et al. (1999), von Mayrhauser et al. (2000) e Wotawa (2016).

As propostas de ferramentas identificadas foram analisadas para se responder a QP1.1. A ferramenta proposta por Memon et al. (2001), chamada *Planning Assisted Tester for grapHical user interface Systems* (PATHS), tem como função a geração de casos de teste para interfaces gráficas. Li et al. (2009) apresentam a *Multiple Fact Files - Interference Progression Planner* (MF-IPP), ferramenta para geração de casos de teste que divide o arquivo de entrada em arquivos menores com o objetivo de ganho de desempenho. Gupta et al. (2007) apresenta a ferramenta *Means-Ends Analysis Graphplan* (MEA-Graphplan), que gera o grafo de planejamento do estado objetivo até o inicial e posteriormente utiliza o algoritmo *Graphplan* para a geração de casos de teste. Respondendo QP1.2, as três ferramentas identificadas não possuem o código aberto.

- **QP2: Qual a fase de teste de software utilizada?**

Para a QP2, as publicações foram analisadas a fim de se identificar a fase de teste em que as propostas foram realizadas. Nenhuma publicação especifica a fase de teste de software empregada em suas propostas. As propostas de Wotawa (2016) e Bozic e Wotawa (2018) referem-se a teste de segurança, logo subentende-se que ocorrem na fase de teste de sistema.

- **QP3: Qual a técnica de teste de software utilizada?**

A QP3 foi respondida com a análise das publicações com o objetivo de identificar a técnica de teste empregada. A proposta de Razavi et al. (2014) emprega a técnica de teste estrutural. Já von Mayrhauser et al. (2000) utilizam a técnica de teste baseado em erros. As demais publicações utilizam a técnica de teste de funcional. Portanto, nota-se que o teste funcional é o mais frequente entre as publicações selecionadas, utilizado em 87% das propostas analisadas.

- **QP4: Qual a técnica de planejamento em IA utilizada?**

Na QP4 foram identificadas cinco técnicas distintas de planejamento em IA. Mraz et al. (1995), Howe et al. (1995), Howe et al. (1997), Scheetz et al. (1999), von Mayrhauser et al. (1999) e von Mayrhauser et al. (2000) utilizam a linguagem ADL (Pednault, 1989) em suas propostas. A técnica HTN (Erol et al., 1994) é utilizada por Memon et al. (2001). Gupta et al. (2007) e Li et al. (2009) utilizam a técnica Análise de Grafo de Planejamento (do inglês, *Planning Graph Analysis*) (Blum e Furst, 1997). A linguagem PDDL (McDermott et al., 1998) é utilizada por Razavi et al. (2014) e Bozic e Wotawa (2018). A linguagem STRIPS (Fikes e Nilsson, 1971) é utilizado por Wotawa (2016). As demais publicações não especificam a técnica de planejamento em IA utilizada.

- **QP5: Qual(is) artefato(s) utilizado(s) para definição do problema de planejamento?**

Na QP5 foram identificados os artefatos utilizados na definição dos problemas de planejamento em IA. Mraz et al. (1995), Howe et al. (1995) e Howe et al. (1997) definem o problema de planejamento a partir de um conjunto de classes da biblioteca *StorageTek Robot Tape Library*. Diagramas de classes em Linguagem de Modelagem Unificada (UML, do inglês *Unified Modeling Language.*) (Fowler et al., 1997) são utilizados por Scheetz et al. (1999) e von Mayrhauser et al. (1999). Já Scheetz et al. (1999) utilizam diagramas de estados UML (Fowler et al., 1997) para a definição do problema.

Em von Mayrhauser et al. (2000) o problema é definido a partir de operadores de mutação. Diagramas de interface (resumo das entradas e saídas dos módulos do sistema) são utilizados por Feather e Smith (2001). Memon et al. (1999) e Memon et al. (2001) utilizam funções da interface do sistema. Artefatos com especificação de componentes (parâmetros de entrada, tempo de processamento, recursos, saída) do sistema são utilizados por Yen et al. (2002). Gupta et al. (2007) definem o problema a partir de máquinas de estados finita que representam os possíveis estados do sistema.

Li et al. (2009) utilizam artefatos contendo características da interface do usuário como botões, parâmetros de entrada e seleção de parâmetros. Razavi et al. (2014) utiliza um histórico de variáveis compartilhadas por programas Java concorrentes e padrões das violações (condições de corrida, atomicidade e ponteiros nulos). Descrição das vulnerabilidades XSS (Grossman et al., 2007) e injeção de SQL (Clarke-Salt, 2009) são utilizadas por Wotawa (2016) e Bozic e Wotawa (2018). As demais publicações realizam a definição do problema de planejamento em IA de acordo com o estado atual do sistema, não utilizando um artefato específico.

- **QP6: Qual o planejador utilizado?**

Na QP6 foram identificados cinco planejadores distintos. Diferentes versões do planejador UCPOP (Penberthy et al., 1992) são utilizadas por Mraz et al. (1995), Howe et al. (1995), Howe et al. (1997), Scheetz et al. (1999), von Mayrhauser et al. (1999) e von Mayrhauser et al. (2000). Memon et al. (1999) utilizam o Planejador de Progressão de Interferência (IPP, do inglês *Interference Progression Planner*) (Koehler et al., 1997). O planejador DS-1 *Planner* (DS1, 2019) é utilizado em Feather e Smith (2001). O planejador *Fast Foward* (FF) (Hoffmann e Nebel, 2001) é empregado na proposta de Razavi et al. (2014). Já Bozic e Wotawa (2018) utilizam o planejador JavaGP (2017). Yen et al. (2002) e Wotawa (2016) não especificam o uso de nenhum planejador. As três publicações restantes contêm proposta e implementação de ferramentas próprias.

- **QP7: Qual(is) a(s) ferramenta(s) de apoio utilizada(s)?**

Na QP7 foram identificadas as seguintes ferramentas de apoio. Mraz et al. (1995), Howe et al. (1995) e Howe et al. (1997) utilizam a ferramenta Sleuth (Von Mayrhauser et al., 1994), utilizada nos experimentos do autores a fim de comparar seus resultados obtidos com a execução do planejador. A interface gráfica do WordPad (2019) é utilizada nos estudos de caso realizados por Memon et al. (1999) e Memon et al. (2001). JPlan (EL-Manzalawy, 2006) é utilizado em Gupta et al. (2007) e IPP (Koehler et al., 1997) em Li et al. (2009). JPlan e IPP planejadores que serviram como base para as ferramentas propostas pelos autores. A ferramenta de teste Penelope (Sorrentino et al., 2010) teve sua estrutura interna modificada com a inclusão de um planejador em Razavi et al. (2014), com o objetivo de ganho de desempenho na geração dos casos de teste. HttpClient (2019), jsoup (2019) e Crawler4j (2019) são utilizados em Bozic e Wotawa (2018). HttpClient faz configuração de mensagens HTTP, jsoup é uma biblioteca Java que analisa as respostas recebidos do sistema sob teste e Crawler4j define a profundidade do rastreamento e o número de páginas Web a serem buscadas.

- **QP8: O quê o plano gerado representa?**

A fim de se responder a QP8, os planos gerados pelas propostas foram analisados. Os planos gerados por Mraz et al. (1995), Howe et al. (1995), Howe et al. (1997), Scheetz et al. (1999), Memon et al. (1999), Memon et al. (2001) e Li et al. (2009) representam casos de teste.

Já o plano gerado por von Mayrhauser et al. (2000) representa casos de teste de recuperação de erros. As demais publicações não descrevem os planos resultantes.

A Tabela A.10 apresenta uma síntese dos dados extraídos na amostra final de publicações. As colunas "Proposta", "Técnica/Fase de teste", "Artefato", "Técnica", "Planejador" e "Apoio" referem-se às questões de pesquisa QP1 a QP7, respectivamente. Dados não contidos nas publicações e obtidos por pesquisa adicional são apresentados entre parênteses. As marcações "-" indicam dados não contidos nas publicações.

Tabela A.10: MSL-Teste - Síntese da extração de dados. Fonte: O autor (2019).

Publicação	**Proposta**	**Técnica/ Fase de teste**	**Artefato**	**Técnica**	**Plane-jador**	**Apoio**
Mraz et al., (1995)	Modelo	(Caixa-preta)	Classes	(ADL)	UCPOP	Sleuth
Howe et al., (1995)	Modelo	(Caixa-preta)	Classes	(ADL)	UCPOP	Sleuth
Howe et al., (1997)	Modelo	Caixa-preta	Classes	(ADL)	UCPOP 2.0	Sleuth
Scheetz et al., (1999)	Modelo	Caixa-preta	Diagramas de classes e estados UML diagram	(ADL)	UCPOP 4.0	-
von Mayrhauser et al., (1999)	Modelo	Caixa-preta	Diagrama de classes UML	(ADL)	UCPOP	-
Memon et al., (1999)	Técnica	(Caixa-preta)	Funções da interface	(ADL)	IPP	Word-Pad
von Mayrhauser et al., (2000)	Técnica	Baseado em erros	Operado-res de mutação	(ADL)	UCPOP 4.0	-
Feather and Smith, (2001)	*Framework*	(Caixa-preta)	Diagra-mas da interface	-	DS-1	-
Memon et al., (2001)	Ferramenta	(Caixa-preta)	Funções da interface	HTN	PATHS	Word-Pad
Yen et al., (2002)	*Framework*	(Caixa-preta)	Descrição de elemen-tos do sistema	-	-	-

Continua na próxima página

Tabela A.10 - *Continua da página anterior*

Publicação	**Proposta**	**Técnica/ Fase de teste**	**Artefato**	**Técnica**	**Plane-jador**	**Apoio**
Gupta et al., (2007)	Ferramenta	(Caixa-preta)	Máquina de estados finita	*Planning Graph Analysis*	MEA-Graphplan	JPlan
Li et al., (2009)	Ferramenta	(Caixa-preta)	Funções da interface	(*Planning Graph Analysis*)	MF-IPP	IPP
Razavi et al., (2014)	*Framework*	(Caixa-branca)	Variáveis e padrões de violação	PDDL	FF	Penelope
Wotawa, (2016)	Técnica	Caixa-preta/ (Sistema)	Descrição de vulnera-bilidades	STRIPS	-	-
Bozic and Wotawa, (2018)	*Framework*	(Caixa-preta)/ (Sistema)	Descrição de vulnera-bilidades	PDDL	JavaGP	HttpClient, jsoup, e (Crawler4j)

APÊNDICE B – MODELAGEM DO DOMÍNIO EM PDDL

```
(define (domain dominio_teste_intrusao)
  (:requirements :typing :fluents)
  (:types
    aplicacao
    estado
    injecao_sql
    xss
    exploit
  )

  (:predicates
    (estado ?s - estado)
    (conexao_estados ?s1 ?s2 - estado)
    (autenticacao_obtida ?ast - aplicacao)
    (crawler_executado ?ast - aplicacao)
    (varredura_executada ?ast - aplicacao)
    (exploracao_executada ?ast - aplicacao)
    (tem_autenticacao ?ast - aplicacao)
    (teste_injecao_sql ?v - injecao_sql)
    (teste_xss ?v - xss)
    (executa_exploit ?e - exploit)
  )

  (:functions (prioridade))

  ;Aut1
  (:action info_autenticacao_wapiti
    :parameters   (?ast - aplicacao)
    :precondition (and (not(crawler_executado ?ast))
                       (not(varredura_executada ?ast))
                       (not(exploracao_executada ?ast))
                       (tem_autenticacao ?ast))

    :effect       (and (autenticacao_obtida ?ast))
  )

  ;Cr1
  (:action crawler_htcap_sem_autenticacao
    :parameters   (?ast - aplicacao
                   ?atual - estado ?proximo - estado)
    :precondition (and (estado ?atual)
                       (conexao_estados ?atual ?proximo)
                       (not(crawler_executado ?ast))
                       (not(varredura_executada ?ast))
```

```
                    (not(exploracao_executada ?ast))
                    (not(tem_autenticacao ?ast)))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (crawler_executado ?ast))
)

;Cr2
(:action crawler_htcap_com_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                    (conexao_estados ?atual ?proximo)
                    (not(crawler_executado ?ast))
                    (not(varredura_executada ?ast))
                    (not(exploracao_executada ?ast))
                    (tem_autenticacao ?ast)
                    (autenticacao_obtida ?ast))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (crawler_executado ?ast))
)

;Varr1
(:action varredura_skipfish_sem_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                    (conexao_estados ?atual ?proximo)
                    (crawler_executado ?ast)
                    (not(varredura_executada ?ast))
                    (not(exploracao_executada ?ast))
                    (not(tem_autenticacao ?ast)))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (varredura_executada ?ast)
                    (increase (prioridade) 3))
)

;Varr2
(:action varredura_arachni_sem_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                    (conexao_estados ?atual ?proximo)
```

```
                    (crawler_executado ?ast)
                    (not(varredura_executada ?ast))
                    (not(exploracao_executada ?ast))
                    (not(tem_autenticacao ?ast)))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (varredura_executada ?ast)
                    (increase (prioridade) 2))
)

;Varr3
(:action varredura_zap_sem_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                    (conexao_estados ?atual ?proximo)
                    (crawler_executado ?ast)
                    (not(varredura_executada ?ast))
                    (not(exploracao_executada ?ast))
                    (not(tem_autenticacao ?ast)))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (varredura_executada ?ast)
                    (increase (prioridade) 1))
)

;Varr4
(:action varredura_skipfish_com_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                    (conexao_estados ?atual ?proximo)
                    (crawler_executado ?ast)
                    (not(varredura_executada ?ast))
                    (not(exploracao_executada ?ast))
                    (tem_autenticacao ?ast)
                    (autenticacao_obtida ?ast))

  :effect       (and (estado ?proximo)
                    (not(estado ?atual))
                    (varredura_executada ?ast)
                    (increase (prioridade) 3))
)

;Varr5
(:action varredura_arachni_com_autenticacao
```

```
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (not(varredura_executada ?ast))
                     (not(exploracao_executada ?ast))
                     (tem_autenticacao ?ast)
                     (autenticacao_obtida ?ast))

  :effect       (and (estado ?proximo)
                     (not(estado ?atual))
                     (varredura_executada ?ast)
                     (increase (prioridade) 2))
)

;Varr6
(:action varredura_zap_com_autenticacao
  :parameters   (?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (not(varredura_executada ?ast))
                     (not(exploracao_executada ?ast))
                     (tem_autenticacao ?ast)
                     (autenticacao_obtida ?ast))

  :effect       (and (estado ?proximo)
                     (not(estado ?atual))
                     (varredura_executada ?ast)
                     (increase (prioridade) 1))
)

;Exp1
(:action exploracao_sqlmap_sem_autenticacao
  :parameters   (?v - injecao_sql ?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (varredura_executada ?ast)
                     (not(exploracao_executada ?ast))
                     (not(tem_autenticacao ?ast))
                     (teste_injecao_sql ?v))

  :effect       (and (estado ?proximo)
                     (not(estado ?atual))
```

```
                    (exploracao_executada ?ast))
)

;Exp2
(:action exploracao_xsser_sem_autenticacao
  :parameters   (?v - xss ?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (varredura_executada ?ast)
                     (not(exploracao_executada ?ast))
                     (not(tem_autenticacao ?ast))
                     (teste_xss ?v))

  :effect       (and (estado ?proximo)
                     (not(estado ?atual))
                     (exploracao_executada ?ast))
)

;Exp3
(:action exploracao_sqlmap_com_autenticacao
  :parameters   (?v - injecao_sql ?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (varredura_executada ?ast)
                     (not(exploracao_executada ?ast))
                     (tem_autenticacao ?ast)
                     (autenticacao_obtida ?ast)
                     (teste_injecao_sql ?v))

  :effect       (and (estado ?proximo)
                     (not(estado ?atual))
                     (exploracao_executada ?ast))
)

;Exp4
(:action exploracao_xsser_com_autenticacao
  :parameters   (?v - xss ?ast - aplicacao
                 ?atual - estado ?proximo - estado)
  :precondition (and (estado ?atual)
                     (conexao_estados ?atual ?proximo)
                     (crawler_executado ?ast)
                     (varredura_executada ?ast)
                     (not(exploracao_executada ?ast))
                     (tem_autenticacao ?ast)
```

```
                        (autenticacao_obtida ?ast)
                        (teste_xss ?v))

    :effect         (and (estado ?proximo)
                        (not(estado ?atual))
                        (exploracao_executada ?ast))
  )

  ;Frw1
  (:action exploracao_metasploit_exploit
    :parameters   (?e - exploit ?ast - aplicacao
                   ?atual - estado ?proximo - estado)
    :precondition (and (estado ?atual)
                        (conexao_estados ?atual ?proximo)
                        (crawler_executado ?ast)
                        (varredura_executada ?ast)
                        (not(exploracao_executada ?ast))
                        (executa_exploit ?e))

    :effect         (and (estado ?proximo)
                        (not(estado ?atual))
                        (exploracao_executada ?ast))
  )

)
```

APÊNDICE C – MÓDULOS DO MÉTODO DE TESTE DE INTRUSÃO

C.1 MÓDULO M1: INSTRUMENTALIZAÇÃO DO TESTE

```
LOGIN_PAGE=(
    ''
    'http://testesseg.c3sl.ufpr.br:3001/WebGoat/login'
    'http://testesseg.c3sl.ufpr.br:3002/login.php'
    'http://testesseg.c3sl.ufpr.br:3003/login.php'
    ''
    ''
    ''
    )
URL_EXCLUDE=(
    ''
    'http://testesseg.c3sl.ufpr.br:3001/WebGoat/logout'
    'http://testesseg.c3sl.ufpr.br:3002/logout.php'
    'http://testesseg.c3sl.ufpr.br:3003/logout.php'
    ''
    'http://testesseg.c3sl.ufpr.br:3005/xvwa/setup/'
    ''
    )
URL=(
    'http://testesseg.c3sl.ufpr.br:3000/cgi-bin/badstore.cgi'
    'http://testesseg.c3sl.ufpr.br:3001/WebGoat/start.mvc'
    'http://testesseg.c3sl.ufpr.br:3002/index.php'
    'http://testesseg.c3sl.ufpr.br:3003/portal.php'
    'http://testesseg.c3sl.ufpr.br:3004/index.php'
    'http://testesseg.c3sl.ufpr.br:3005/xvwa/'
    'http://testesseg.c3sl.ufpr.br:3006/btslab'
    )
NAME=(
    'badstore'
    'webgoat'
    'DVWA'
    'bWAPP'
    'Mutillidae'
    'xvwa'
    'btslab'
    )
LOGIN=(
    '0'
    '1'
    '1'
    '1'
    '0'
```

```
    '0'
    '0'
    )
for i in {0..6}; do
    if [[ ${LOGIN[$i]} -eq 1 ]];then
        #getcookie
        echo "${NAME[$i]}"
        /home/tester/tut/wapiti/bin/wapiti-getcookie
            -u ${LOGIN_PAGE[$i]} -c ${NAME[$i]}.json |
            tee ${NAME[$i]}.command

        tail -2 ${NAME[$i]}.command
        cookie=$(tail -n 3 ${NAME[$i]}.command |
                 grep -Eo "([a-zA-Z])+=[[:alnum:]]{4,}")

        rm ${NAME[$i]}.command
        echo "Cookie= " $cookie
        python /home/tester/tut/htcap-1.0.1/htcap.py
            crawl  -v -m active -d domain -x ${URL_EXCLUDE[$i]}
            -c $cookie ${URL[$i]} ${NAME[$i]}.db

    else
        echo "${NAME[$i]}"
        python /home/tester/tut/htcap-1.0.1/htcap.py  crawl
             -d domain -x '${URL_EXCLUDE[$i]}'
             ${URL[$i]} ${NAME[$i]}.db

    fi
        #sqlite3 ${NAME[$i]}.db "select url from request;"
        | cut -d'?' -f1 | uniq | grep "testesseg"
done
```

C.2 MÓDULO M2: VARREDURA DE APLICAÇÃO

```
for file in $(ls *.db); do
    echo $file;
    urls=$(sqlite3 $file "select url from request;"
    | cut -d'?' -f1 | uniq | grep "testesseg");

    for url in $urls; do
        if [ "$1" == "-v" ]; then
            echo $url;
        else
        echo $url;
            zap-cli open-url $url
            zap-cli active-scan --scanners xss,sqli
                    --recursive $url
            zap-cli alerts
```

```
        fi
    done
done
```

C.3 MÓDULO M3: EXPLORAÇÃO DE VULNERABILIDADES

```
tail -n +4 scan.md | grep -v '^\+' | while read line; do
    table=$(echo $line |cut -d'|' -f4)
    url=$(echo $line |cut -d'|' -f5)
    if [[ table -eq 79 ]]; then
        url=$(echo $url | cut -d'=' -f1)
        echo "XSS in URL: $url"
        python2 /home/tester/tut/xsser/xsser/xsser  -u "$url"
    fi

    if [[ table -eq 89 ]]; then
        url=$(echo $url | cut -d'=' -f1)"=*"
        echo "SQLI in URL: $url"
        python3 /home/tester/tut/sqlmap-dev/sqlmap.py
        --level=5 --risk=3 -u "$url" --batch
    fi
done
```